Méthodologie et architecture adaptative pour le placement efficace de tâches matérielles de tailles variables sur des partitions reconfigurables

Nicolas Marques

U.F.R. Faculté des Sciences et Technologies
École doctorale Informatique Automatique Électronique Électrotechnique Mathématiques
Département de formation doctorale Électronique Électrotechnique

Méthodologie et architecture adaptative pour le placement efficace de tâches matérielles de tailles variables sur des partitions reconfigurables

THÈSE

présentée et soutenue publiquement le 26 novembre 2012

pour l'obtention du

Doctorat de l'Université de Lorraine

(Spécialité systèmes électroniques)

par

Nicolas MARQUES

Composition du jury

Président :	M. Patrick GARDA, Professeur, Université Pierre et Marie Curie, Paris 6
Rapporteurs :	M. Christophe JEGO, Professeur, Université de Bordeaux , Bordeaux M. El-Bay BOURENNANE, Professeur, Université de Bourgogne, Dijon
Examinateurs :	M. Hassan RABAH, Professeur, Université de Lorraine, Nancy - Directeur de thèse M. Serge WEBER, Professeur, Université de Lorraine, Nancy - Co-directeur de thèse

Laboratoire d'Instrumentation Électronique de Nancy
Faculté des Sciences - 54506 Vandoeuvre-lès-Nancy

Remerciements

Je remercie en tout premier lieu mes directeurs de thèse, M. Hassan RABAH et M. Serge WEBER, pour la confiance qu'ils m'ont témoignée tout au long de ces années de travail, pour leurs participations, pour leurs encouragements quotidiens et pour leurs précieux conseils au niveau scientifique.

Je voudrais exprimer ma gratitude à toutes les personnes qui m'ont fait l'honneur de participer à ce jury de thèse M. Patrick Garda, professeur à l'Université Pierre et Marie Curie de Paris, M. Christophe Jégo, professeur à l'Université de Bordeaux et M. El-Bay Bourennane, professeur à l'Université de Bourgogne, pour l'attention qu'ils ont accordé à la lecture de ce mémoire de thèse et pour avoir bien voulu en être les rapporteurs.

Mes remerciements les plus chaleureux à tous les membres du LIEN pour leur aide ou les conseils qu'ils ont pu m'apporter. Avec vous, la bonne humeur est indissociable du bon café, du thé et autres douceurs. Tout particulièrement à Eric Dabellani mon collègue sur le projet ARDMAHN, Patrice pour son soutien et son sens de l'humour, Yves et Slaviša pour leurs conseils sur la rédaction et Christine Daudens pour ses conseils administratifs.

Mes remerciements vont également à ma famille, en particulier mes parents, ma sœur, mon frère, Valérie et Matthieu. Je remercie aussi Fatima, Murielle et Jean-Paul pour leur soutien. Enfin, je me permet de terminer ces remerciements par une attention particulière à mes amis Deb, Amélie, Céline, Thomas, Jérémy, Renaud et Morgane pour leur soutien et leur " Et cette thèse, ca avance ? " mais également pour leur " Tu vas trouver quelque chose ? ".

Nicolas MARQUES
Novembre 2012

« Celui qui n'a pas d'objectifs ne risque pas de les atteindre »
Sun Tzu

Je dédie ce travail à mes parents :
Maria et Armando

Table des matières

Publications personnelles

Table des figures

Liste des tableaux

Introduction générale

1 Contexte

Avec la convergence de l'électronique, de l'informatique et des télécommunications, les technologies de l'information occupent une place préférentielle dans notre quotidien. En effet, l'évolution des systèmes embarqués et des systèmes sur puces permettent de développer des fonctions de plus en plus complexes sans négliger la vitesse de traitement et la consommation d'énergie. Les circuits intégrés spécifiques à une application appelés ASIC (Application-Specific Integrated Circuit) représentent une solution optimale sur le plan du taux d'intégration au détriment d'un coût de développement et de fabrication important, et d'un degré de flexibilité loin d'être idéal. Par ailleurs, avec la réduction de l'échelle de gravure des circuits, la consommation statique augmente considérablement ce qui devient un problème majeur pour les applications nomades. Les ASICs atteignent une nouvelle limite et ils ne sont viables que pour certaines applications dédiées. Ainsi toute modification ou mise à jour d'un ASIC se soldera par la fabrication d'un nouveau circuit. Or, les systèmes électroniques embarqués ont besoin de circuits capables de répondre aux exigences de coût, de performance, de flexibilité et de consommation. Les circuits logiques programmables de type FPGA (Field-Programmable Gate Array) sont une solution qui dispose de performances acceptables et d'un temps de mise en œuvre réduit par rapport aux ASICs. De plus, avec la présence de technologies reconfigurables de type FPGA il est possible d'atteindre une grande flexibilité. Toutefois, l'inconvénient majeur des FPGA est la surface logique nécessaire à la réalisation d'une application sur FPGA et leurs performances qui sont équivalentes à environ un tiers de celles des ASICs mais les performances atteintes par les FPGA restent supérieures à celles des microprocesseurs et leur flexibilité d'adaptation est importante par rapport à celles des ASICs. C'est dans ce contexte que les FPGA reconfigurables apportent de nouvelles solutions pour la conception de systèmes.

De nos jours, les technologies reconfigurables de types FPGA sont perçues comme une nouvelle alternative pour la réalisation de systèmes de traitement numérique de l'information. L'idée maîtresse de cette technologie est d'adapter un système en fonction du besoin grâce à la modification dynamique de son architecture pendant l'exécution d'un calcul. Ce concept est déjà relativement ancien [EV62], mais ce n'est que très récemment que la reconfiguration dynamique a pu être mise en œuvre sur les FPGA [CH02] [CHW00]. Les premières utilisations de la reconfiguration dynamique étaient principalement axées sur l'accélération de certains traitements qui se prêtaient bien à la parallélisation spatiale [CH02]. Elle permet d'exécuter un algorithme qui réalise en même temps l'ensemble des opérations. Par la suite, des méthodologies de partitionnement ont également été proposées [CHW00]. C'est dans ce contexte que les technologies reconfigurables de types FPGA permettent une flexibilité grâce à la reconfiguration dynamique (globales ou partielles) de la puce tout en conservant de bonnes performances.

Grâce à la reconfiguration dynamique partielle, on peut supprimer certaines tâches matérielles du FPGA et charger de nouvelles tâches à leurs places. L'implementation de ces nouvelles tâches matérielles sur des régions reconfigurables permet d'augmenter l'adaptabilité du système. Les tâches matérielles doivent être entièrement portables afin d'être facilement insérées dans n'importe quel système, mais elles peuvent être de tailles différentes. Par conséquent, le placement des tâches matérielles de tailles différentes sur des régions reconfigurables doit être réalisé au mieux pour optimiser l'utilisation des ressources du FPGA.

2 Placement des tâches matérielles

Le placement des tâches matérielles sur un FPGA est important dans la conception des systèmes reconfigurables auto-adaptatifs. Un mauvais placement impliquera une utilisation sous-optimale des ressources et par conséquent une diminution des performances et de l'efficacité du système. Dans le cas où la conception est de type reconfigurable, ce placement devient très complexe. Le placement des tâches doit être réalisé sur des régions reconfigurables en respectant leur temps d'exécution. Il existe deux approches pour traiter ce problème : le placement pendant la phase de conception et le placement au cours de l'exécution du FPGA. La première approche passe par une modélisation des tâches pendant la phase de développement du FPGA, et conduit à définir l'ordre d'exécution et les régions où seront placées les tâches [DP05] [DP06]. Cette approche bien qu'efficace interdit l'adaptation à des événements non prévus dans la phase de conception, car cette méthode fige l'architecture et le placement des tâches matérielles. Dans la seconde approche, les modèles de placement pendant l'exécu-

tion utilisent un ordonnanceur en ligne qui permet de définir à quel moment et/ou seront placé les tâches matérielles [BKS00] [CRGM11]. Si l'ordonnanceur ne parvient pas à trouver un espace libre sur le composant pour placer les tâches matérielles, il décide alors soit d'exécuter la tâche plus tard ou de la rejeter. La majorité des travaux existants sur le placement portent sur la recherche et l'amélioration des performances des algorithmes de placement. Cependant, la problématique sous-jacente du partitionnement physique des FPGA et de la génération efficace des régions reconfigurables pour accueillir des tâches matérielles de différentes tailles, est très peut explorée.

3 Partitionnement du FPGA

Les travaux sur les modèles et les méthodes de placement permettent de définir l'ordre d'exécution et les régions où seront placées les tâches matérielles grâce à des algorithmes de traitement (ordonnanceur et placeur) exécutés par des processeurs logiciels (figure 1).

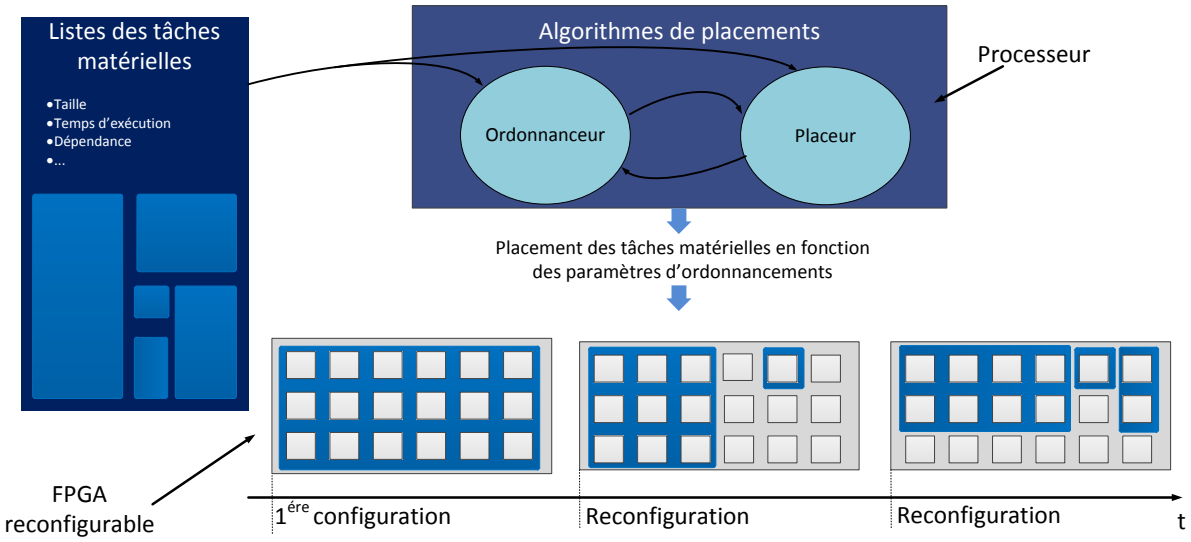


FIGURE 1 – Principe du placement des tâches matérielles sur un FPGA.

Il est important d'ajouter un support matériel qui permet de réaliser le placement des tâches et la communication entre elle. Un placement efficace des tâches matérielles sur un FPGA reconfigurable peut être réalisé grâce à une modélisation de la structure du FPGA. Les tâches matérielles sont définies comme des tâches matérielles reconfigurables ou non qui doivent être placées sur le FPGA. La problématique du partitionnement [DE97] [WK02] permet de déterminer un site d'allocation potentiel pour des tâches matérielles reconfigurables en prenant en compte les ressources nécessaires à l'implémentation et celles disponibles sur le circuit. La plupart des

travaux définissent la taille des tâches matérielles par les ressources CLBs qu'elles utilisent sur le FPGA et modélisent le FPGA par une matrice de blocs CLBs (x, y) [GGJ+04] [HSB06a]. Chaque cellule de la matrice représente un bloc CLB. Ces cellules sont ensuite pondérées en fonction de leur état (libre ou occupée). En revanche, les ressources comme les RAM, les blocs DSP et autres ne sont généralement pas pris en compte. Par ailleurs, le postulat que les FPGA ont une structure régulière et homogène dans tout le circuit n'est plus vrai dans les circuits de dernière génération en particulier si les éléments comme les RAM et les blocs DSP sont pris en compte. D'autres auteurs [XDN09] [BMB10] proposent des méthodes prenant en comptent tous les types de cellules (RAM, DSP, ...) mais ils ne prennent pas en compte les problèmes de communication et les contraintes de la reconfiguration dynamique. Le partitionnement des régions reconfigurables introduit aussi une nouvelle problématique au niveau des interfaces de communication entre les partitions reconfigurables. Quand une région reconfigurable est divisée en plusieurs partitions, chaque partition doit être capable de communiquer avec les autres partitions ou avec la région reconfigurable. Cette communication permet le bon acheminement des données et elle doit être flexible pour s'adapter au changement de tâches matérielles. Cette contrainte de communication est loin d'être évidente et constitue une limitation majeure au développement de la reconfiguration dynamique dans les applications industrielles.

4 Contribution

Le partitionnement de la région reconfigurable d'un FPGA consiste à définir des partitions de tailles fixes capables d'accueillir des tâches matérielles et d'assurer la communication avec le reste du système. L'utilisation de partitions fixes pose la problématique de l'efficacité en surface dans le cas où les tâches matérielles sont de tailles différentes. L'efficacité en surface (Es) peut être définie comme une métrique (équation 1) qui correspond au rapport entre la taille de la tâche matérielle (TTM) et la taille de la région reconfigurable (TRR).

$$Es = TTM/TRR \tag{1}$$

L'efficacité maximale correspond à Es=100%, indiquant qu'une tâche matérielle occupe la totalité de la partition. Dans le cas d'une application composée de plusieurs tâches de tailles différentes, leur placement sur une partition reconfigurable de taille fixe induirait une dégradation de l'efficacité en surface. L'optimisation de l'efficacité pour des tâches matérielles de taille différente représente donc une problématique importante dans la conception des systèmes adaptatifs. Avec la technologie actuelle ce but ne peut être atteint, car les partitions reconfigurables sont figées au moment de la conception.

La contribution de cette thèse est le développement d'une méthodologie et d'une couche intermédiaire entre le FPGA et l'application permettant le placement efficace des tâches matérielles de tailles différentes sur des partitions reconfigurables fixes. Les travaux menés et présentés dans cette thèse ont été entrepris dans le cadre du projet Architecture Reconfigurable Dynamiquement et Méthodologie pour l'Autoadaptation en Home Networking (ARDMAHN) [Ard]. Ce travail bénéficie d'une aide de l'Agence Nationale de la Recherche portant la référence ANR-09-SEGI-001. L'objectif de ces travaux de recherche sera d'obtenir une architecture reconfigurable qui s'adapte à son environnement pour les passerelles multimédias.

5 Plan du manuscrit

Le but des travaux de recherche présentés dans ce manuscrit est d'apporter des solutions architecturales innovantes pour améliorer la reconfiguration dynamique. Plus précisément, ces travaux de thèse visent à développer une méthode pour augmenter la flexibilité et donc l'adaptabilité des architectures reconfigurables. Cette thèse est structurée de la maniére suivante.

Le **premier chapitre** présente l'adaptation matérielle à base de FPGA. Dans ce chapitre, nous rappelons les principales caractéristiques des FPGA et des architectures de système à base de FPGA. Puis, la reconfiguration dynamique des FPGA sera présentée

Le **second chapitre** sera entièrement dédié aux techniques de reconfiguration des FPGA. Après une introduction et un rappel sur les terminologies, une présentation des techniques de reconfiguration des FPGA sera faite. Pour finir, nous présenterons des contraintes de mise en œuvre de la reconfiguration.

Le **troisième chapitre** présente les principes de l'adaptation des partitions aux modules reconfigurables. Après un état de l'art de ce domaine, nous pourrons conclure sur les points à améliorer.

Le **quatrième chapitre** présente la méthode et l'architecture proposées pour répondre aux contraintes de placement des tâches matérielles de tailles différentes sur des partitions reconfigurables. Dans un premier temps, les besoins pour obtenir ce type d'architecture seront définis. Ensuite, une méthode sera proposée pour la conception de l'architecture.

Le **cinquième chapitre** présente une étude et une validation de la méthode. Nous présenterons les architectures cibles puis une étude et une validation de la méthode de conception seront présentées. Pour finir, une évaluation du partitionnement d'une architecture sera présentée.

La **dernière partie** tiendra lieu de conclusion finale. Nous discuterons et dresserons le bilan du travail de thèse en présentant les apports proposés et en dégageant les perspectives ouvertes par ces travaux.

Chapitre 1

L'adaptation matérielle à base de FPGA reconfigurable

Sommaire

1.1 Introduction

La technologie FPGA actuellement disponible permet de construire des systèmes qui sont configurables et/ou reconfigurables. L'idée commune à tous les circuits configurables est de proposer, non pas une fonction figée dont le champ d'application est le plus large possible, mais une structure adaptable. La reconfiguration consiste à modifier en cours de fonctionnement l'architecture matérielle, cela permet de concevoir des circuits intelligents qui peuvent s'auto adapter à leur environnement.

La suite de ce chapitre est organisée de la manière suivante. Dans une première partie, nous rappelons les principales caractéristiques des FPGA et des architectures de système à base de FPGA. Ensuite, nous présenterons la reconfiguration et les différents modes de fonctionnement.

1.2 Présentation des FPGA

1.2.1 Introduction aux technologies FPGA

Les FPGA (field programmable gate array) sont inventés par la société Xilinx en 1985. Xilinx fut précurseur du domaine en lançant le premier circuit FPGA commercial, le XC2000. Ce composant avait une capacité maximum de 1500 portes logiques. La technologie utilisée était alors une technologie aluminium à 2 micro-mètre avec 2 niveaux de métallisation. Xilinx ne sera suivi qu'un peu plus tard, et jamais lâchée, par son plus sérieux concurrent Altera qui lança en 1992 la famille de FPGA FLEX 8000 dont la capacité maximum atteignait 15 000 portes logiques.

Les FPGA se situent entre les réseaux logiques programmables et les circuits logiques près diffusés. Les réseaux logiques programmables sont des composants qui ne nécessitent aucune étape technologique supplémentaire pour être personnalisés, ce sont des circuits standards, programmables par l'utilisateur grâce aux différents outils de développement et qui incluent un grand nombre de solutions basées sur les variantes de l'architecture des portes ET et OU. Les prés diffusés sont des circuits intégrés basés sur l'utilisation des réseaux de cellules dont les blocs ont été préalablement diffusés, il faut créer les connexions entre ces blocs. Les FPGA combinent donc à la fois la souplesse de la programmation des réseaux logiques programmables et les performances des circuits prés diffuses. En d'autres termes le FPGA permet d'avoir une architecture conçue sur mesure à haute densité dans un circuit électronique, avec la possibilité de modifier cette architecture quand des nouvelles applications apparaissent. Les langages

HDL comme le VHDL ou le Verilog sont utilisés pour décrire les fonctionnalités qui seront implémentées sur le composant, puis la description matérielle est traduite dans un fichier de configuration pour le FPGA cible. La description matérielle peut être aussi utilisée pour la description d'un composant ASIC.

En 2000 et 2001, les deux concurrents Xilinx et Altera ont franchi une nouvelle étape au niveau de la densité d'intégration en proposant respectivement leurs circuits Virtex et Apex-II dont les capacités maximums avoisinaient les 4 millions de portes logiques équivalentes avec de plus l'introduction de larges bancs de mémoires embarquées. Aujourd'hui, les fréquences de fonctionnement de ces circuits sont de l'ordre de quelques centaines de Méga Hertz (ces dernières sont en réalité très dépendantes de l'application). Bien que ces valeurs soient relativement réduites par rapport aux ASICs, elles sont suffisantes pour une très large majorité d'applications actuelles. À partir des années 2000, les capacités des FPGA ont permis d'offrir aux concepteurs une solution supplémentaire de réalisation pour une majorité d'applications. De plus, les outils de mise en oeuvre des FPGA ont évolué et bien qu'encore pénalisants lors de la conception, ils permettent la réalisation rapide d'applications complexes.

La flexibilité du FPGA permet de changer sa description matérielle en fonction de l'application, c'est le facteur principal qui a déterminé la popularité des composants FPGA. De plus, les FPGA peuvent être couplés avec un processeur à usage général. Ainsi la section la plus exigeante du logiciel peut être traduite sur la partie matérielle qui permet d'accélérer l'exécution du programme. Ce qui donne des accélérations notables dans l'exécution, en particulier lorsque le programme exécuté en série sur un processeur peut être traduire en matériel pour exploiter le parallélisme de l'algorithme. Par conséquent, les FPGA permettent d'obtenir des performances de calcul importantes. De plus, le nombre de portes logiques sur les FPGA les plus récents permet l'implémentation complète d'un système sur puce (SoC - System On Chip).

1.2.2 L'architecture des FPGA

Cette partie présente les caractéristiques des FPGA, en particulier pour clarifier la façon dont ils peuvent mettre en œuvre la personnalisation du matériel via leur reconfiguration. Bien qu'il existe actuellement plusieurs fabricants de circuits FPGA, dont Xilinx [Xila] et Altera [Alt] qui sont les plus connus, nous décrirons l'architecture des FPGA à partir des circuits Xilinx que nous avons employés dans la suite de notre étude. L'architecture, retenue par Xilinx (figure 1.1), se présente sous la forme de deux couches distinctes, la première est la couche appelée circuit configurable et la deuxième est une couche de réseau mémoire SRAM.

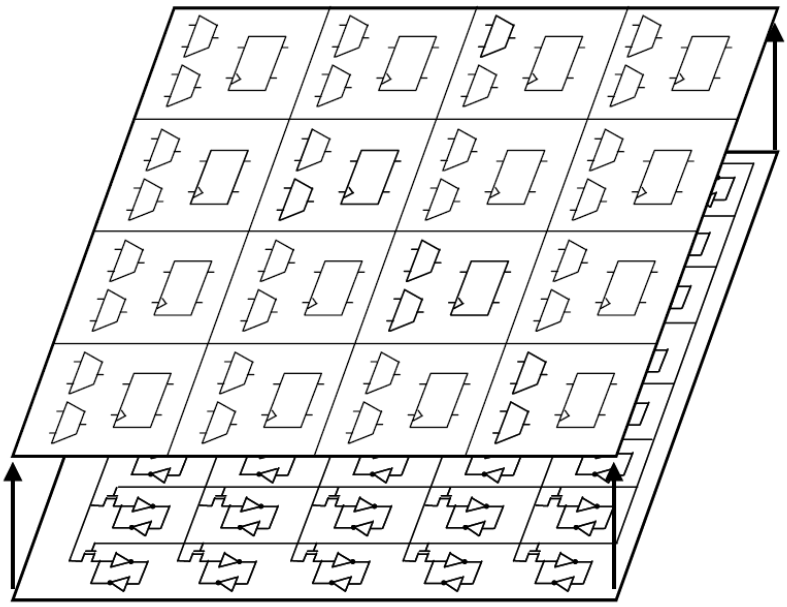


FIGURE 1.1 – Architecture FPGA retenue par Xilinx, la première est la couche appelée circuit configurable et la deuxième est une couche de réseau mémoire SRAM

La première couche (circuit configurable) est constituée d'une matrice de blocs logiques configurables (CLB - Configurable Logic Bloc). Les CLB permettent de réaliser des fonctions séquentielles et combinatoires. Autour de ces blocs logiques configurables, nous trouvons les blocs entrées/sorties (IOB - Input Output Bloc)). Ils permettent de gérer les entrées-sorties pour réaliser l'interface avec les modules extérieurs.

La seconde couche est un réseau de mémoire SRAM qui permet la programmation du circuit FPGA. La programmation est réalisée en appliquant les potentiels adéquats sur la grille de certains transistors à effet de champ pour interconnecter les éléments des CLB et des IOB afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont mémorisés dans le réseau de mémoire SRAM. Un dispositif interne au FPGA permet à chaque mise sous tension de charger les SRAM internes à partir de la ROM externe où est stockée la configuration du FPGA.

1.2.3 Architecture interne d'un FPGA

Les circuits FPGA du fabricant Xilinx utilisent deux types de cellules de base, les cellules d'entrées/sorties appelées IOB et les cellules logiques appelées CLB. Ces différentes cellules

sont reliées entre elles par un réseau d'interconnexions configurable. Donc les trois blocs principaux (figure 1.2) sont les blocs logiques configurables, les blocs d'entrées/sorties et les ressources de communications.

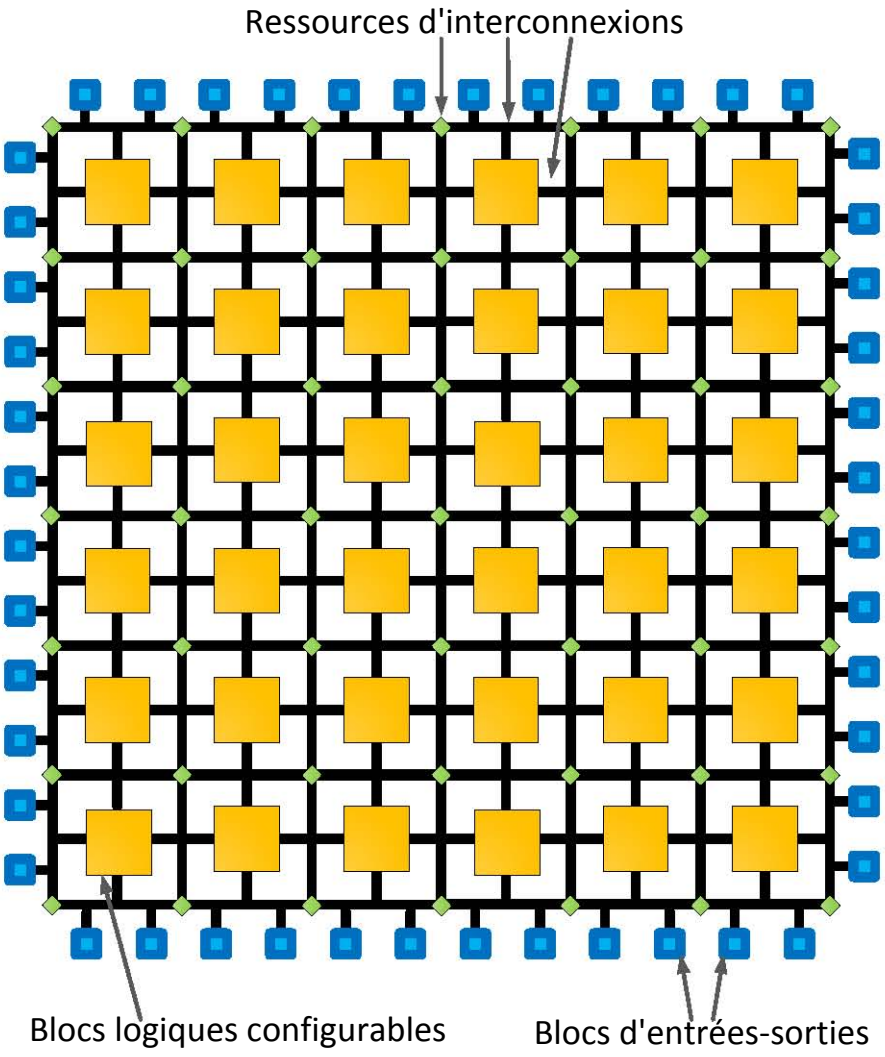


FIGURE 1.2 – Architecture interne d'un FPGA.

1.2.4 Blocs principaux

1.2.4.1 Les blocs logiques configurables (CLB)

Les blocs logiques configurables sont les principaux éléments d'un FPGA. Ils peuvent avoir un ou plusieurs générateurs de fonctions réalisées avec des tables de correspondance (LUT - look-up tables) qui peuvent mettre en œuvre une logique arbitraire en fonction de leur configuration. Autour d'une LUT, il y a une logique d'interconnexion qui permet les liaisons à desti-

nation et vers la LUT. Elle est mise en œuvre à l'aide de portes logiques et de multiplexeurs. Pendant le processus de configuration d'un FPGA, les mémoires des LUT sont écrites pour y implémenter une fonction, et la logique qui l'entoure est configurée pour router correctement les signaux afin de construire un système complexe. Il existe différents types de CLB en fonction du FPGA utilisé. Pour Xilinx le bloc logique configurable FPGA est appelé slice. L'architecture du slice (avec des modifications mineures) est utilisée dans toutes les familles FPGA Virtex et toutes les familles de FPGA Spartan. Un slice Virtex-4 (figure 1.3) est composé de deux LUT4 à 4 entrées (Look-Up Tables). Ces LUT4 peuvent mettre en œuvre une fonction booléenne (une LUT est marquée "F", l'autre est marqué "G"). Il y aussi deux multiplexeurs contrôlés par l'utilisateur (MUXF5 et MUXFX). MUXF5 peut être utilisé pour combiner les sorties des LUT4. MUXFX est utilisé pour combiner les sorties de MUXF5 et MUXFX à partir d'une autre slice. Le bloc arithmétique et logique est composé de deux additionneurs sur 1 bit, d'une chaine de retenue et deux portes ET dédiées à la multiplication. Pour finir, nous avons deux registres sur 1 bit. L'entrée de ces registres est sélectionnée par les multiplexeurs YMUX et XMUX. Notez que ces multiplexeurs ne sont pas contrôlés par l'utilisateur : le chemin est sélectionné lors de la configuration du FPGA. Le schéma simplifié d'un slice Virtex-4 est présenté ci-dessous. Les slices pour les Virtex 5 et 6 sont légèrement différentes (figure 1.4), mais leurs principes de fonctionnement restent similaires au slice du Virtex 4. Il est composé de 4 LUT et 4 registres par slice. Les LUT peuvent être configurés en mode 6 entrées avec 1 bit de sortie ou 5 entrées avec 2 bits de sortie ou encore 4 entrées avec 1 bit de sortie.

1.2.4.2 Les blocs d'entrée-sortie (IOB)

Les blocs d'entrée-sortie permettent l'interconnexion de la logique interne aux ports d'entrées et de sorties du FPGA. Les IOB ont leur propre mémoire de configuration, elle stocke les standards de tension et la direction des ports. Ces blocs sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé (haute impédance).

1.2.4.3 Les ressources d'interconnexion

Les ressources d'interconnexion au sein d'un FPGA permettent la connexion arbitraire des CLB et des IOB. Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent

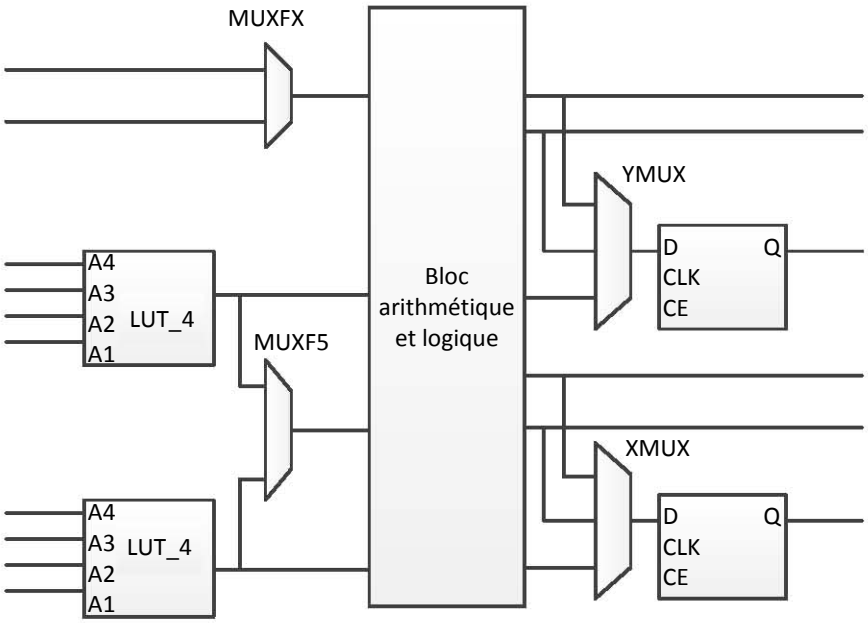


FIGURE 1.3 – Architecture d'un slice Virtex 4.

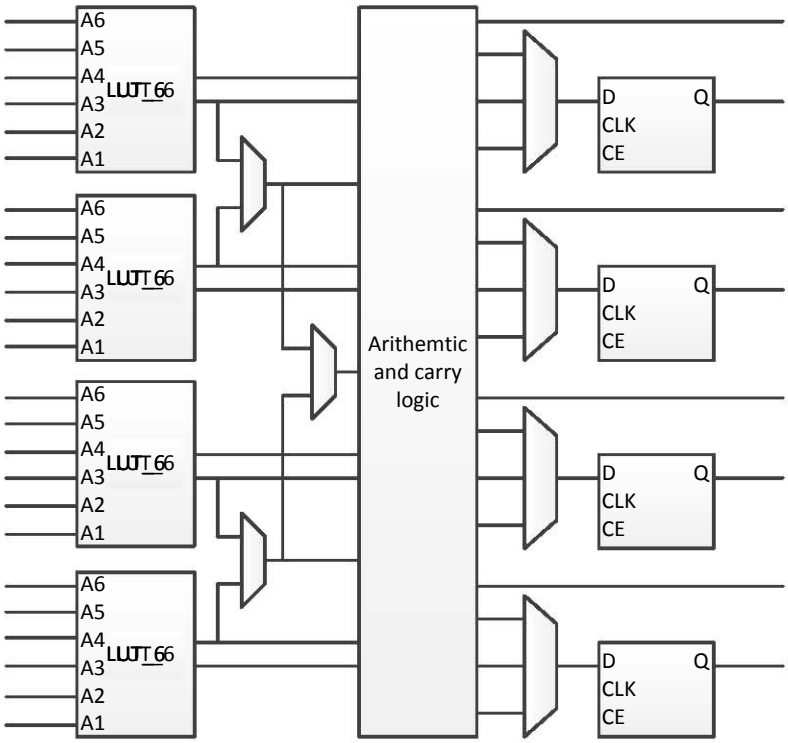


FIGURE 1.4 – Architecture d'un slice Virtex 5

les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible. Pour parvenir à cet objectif, Xilinx propose trois sortes d'interconnexions selon la longueur et la destination des liaisons comme le montre la (figure 1.5).

- Les interconnexions directes : Ces interconnexions permettent l'établissement des liaisons entre les CLB et les IOB. Il est possible aussi de connecter directement certaines entrées d'un CLB aux sorties d'un autre.
- Les longues lignes : Ce sont de longs segments métallisés parcourant toute la longueur et la largeur du FPGA, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible. De plus, ces longues lignes permettent d'éviter la multiplicité des points d'interconnexion.
- Les matrices d'interconnexion : Ce sont des aiguilleurs situés à chaque intersection. Leur rôle est de raccorder les longues lignes entre elles selon diverses configurations. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre CLB sur le FPGA pour assurer la communication des signaux. Pour éviter l'affaiblissement des signaux traversant les longues lignes, des buffers sont implantés dans chaque matrice d'interconnexion.

Les trois blocs présentés jusqu'ici sont interconnectés ensemble dans le dispositif pour créer une infrastructure de communication composée d'un réseaux de communication et d'IOBs autour des CLBs. Des cellules de mémoire liées à chaque bloc détiennent les caractéristiques principales, de telle sorte que les interconnexions entre l'infrastructure de communication, les normes de tension des entrées-sorties d'un IOB, et les équations soient commandées par des valeurs particulières stockées dans une mémoire. Toutes ces configurations sont stockées dans des SRAM qui sont volatiles : lorsque le composant est mis sous tension, toute sa configuration est perdue et il doit être redémarré avec une nouvelle configuration. Habituellement, une machine externe se charge de télécharger la configuration sur le FPGA via une de ses interfaces de configuration, et envoie une commande de démarrage pour signaler que la configuration a eu lieu. Certaines cartes ont une mémoire ROM d'intégrée. Elle permet de stocker la configuration, de sorte qu'elle puisse ensuite être téléchargée sur le FPGA. Dans ce cas, les données de configuration sont copiées sur la mémoire SRAM de configuration du FPGA au démarrage de celui-ci.

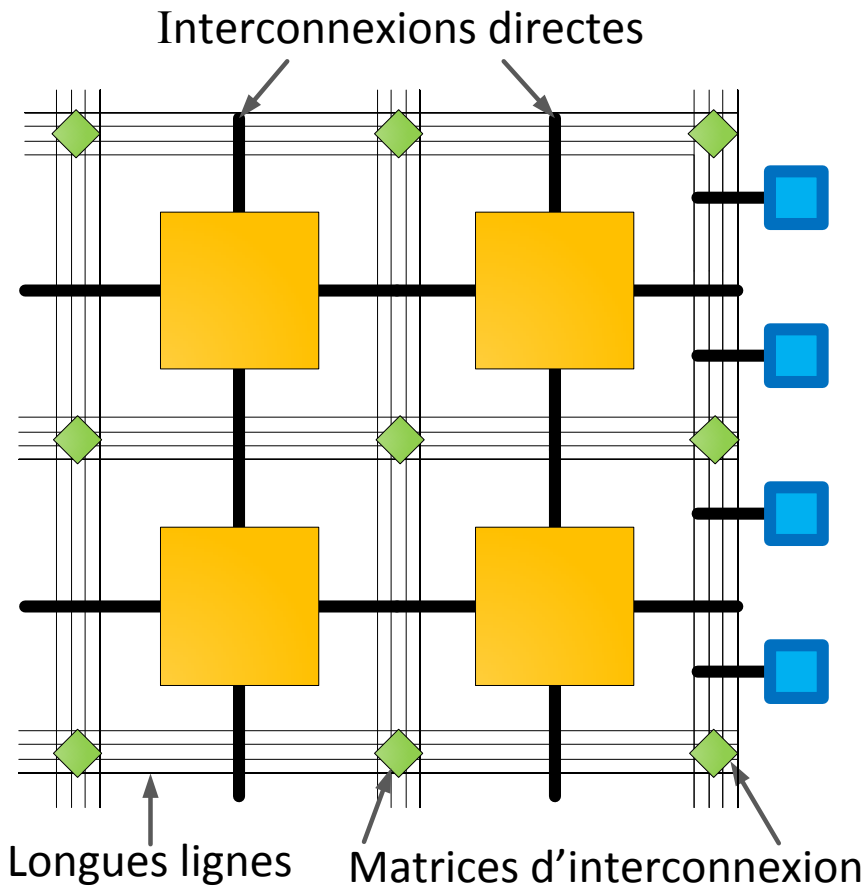


FIGURE 1.5 – Interconnexion interne d'un FPGA.

1.2.5 Bloc à usage spécifique

En plus des trois blocs décrits dans le paragraphe précédent, les FPGA possèdent souvent ce que l'on appelle des ressources additionnelles. La composition détaillée d'un circuit FPGA (figure 1.6), illustre la disposition de ces blocs additionnels sur un FPGA. Le fonctionnement et le placement de ces blocs diffèrent d'une référence de FPGA à une autre.

1.2.5.1 Les blocs RAM (BRAM)

Les blocs RAM sont des mémoires définies par l'utilisateur, embarquées sur le circuit intégré FPGA, elles servent à stocker des ensembles de données. En fonction de la catégorie de FPGA, la mémoire RAM embarquée est configurable en blocs de 16 ou de 32 kilo-octets. Les mémoires RAM sont de type double ports, elles permettent une écriture et une lecture indépendante sur chaque port avec une horloge différente. Ceci est très utile, le composant peut produire (écrire) des données à une fréquence différente d'un autre composant qui consomme (lire) les

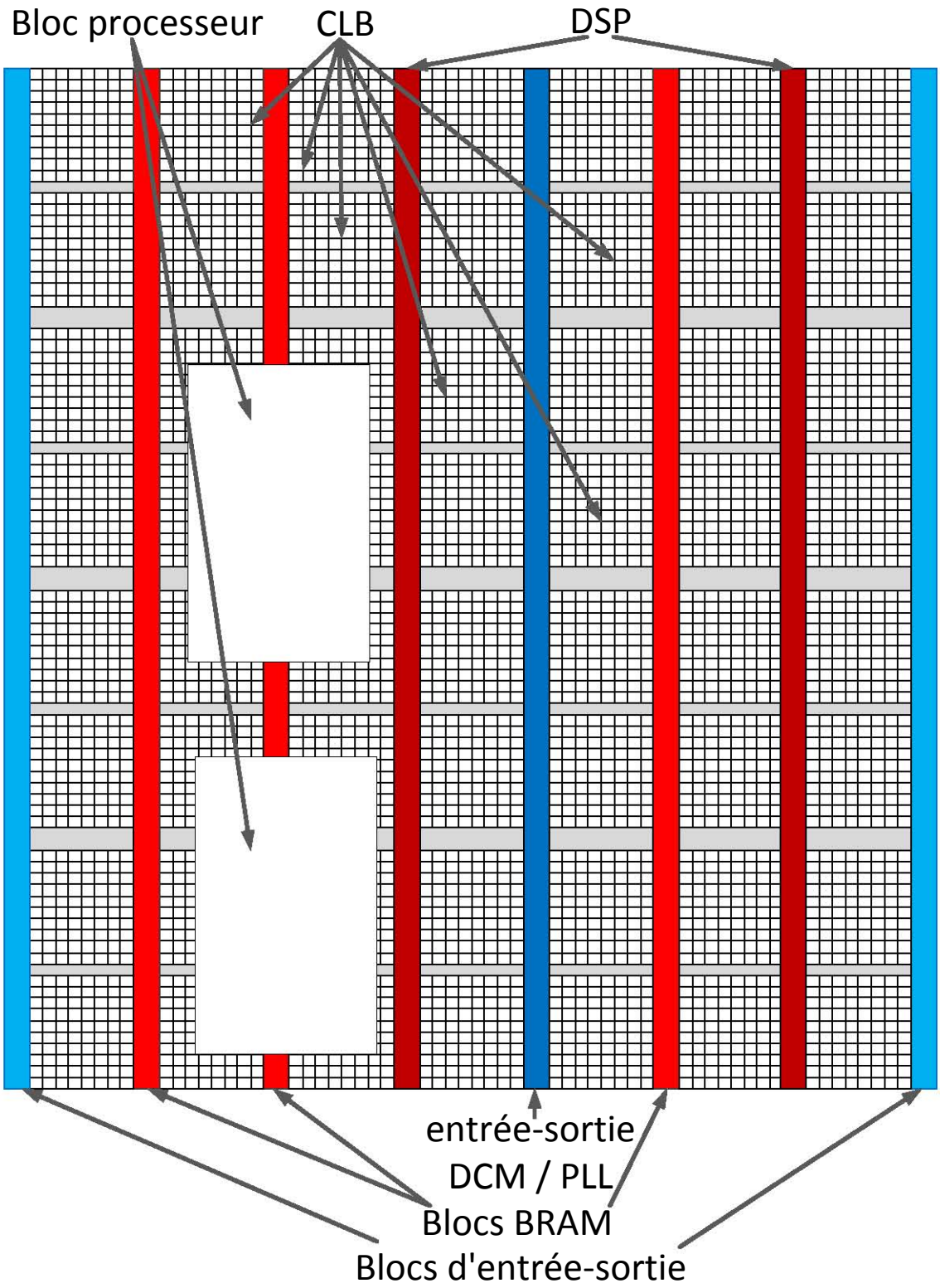


FIGURE 1.6 – Ressources d'un circuit FPGA.

données.

1.2.5.2 Digital Signal Processing (DSP)

Les « Digital Signal Processing » sont des blocs qui permettent des conceptions plus complexes, qui peuvent consister soit en traitement numérique du signal ou seulement certains assortiments de multiplication, addition et soustraction. Comme pour les BRAM, il est possible de mettre en œuvre ces blocs grâce au CLB, mais il est plus efficace en termes de performances, et de consommation d'énergie d'intégrer plusieurs de ces composants au sein du FPGA. Un bloc DSP permet de réaliser un multiplicateur, un accumulateur, un additionneur, et des opérations logiques (AND, OR, NOT, et NAND) sur un bit. Il est possible de combiner les blocs DSP pour effectuer des opérations plus importantes, telles que l'addition avec virgule flottante simple, la soustraction, la multiplication, la division, et la racine carrée. Le nombre de blocs DSP est dépendant du dispositif.

1.2.5.3 Les processeurs embarqués

Les processeurs embarqués enfouis sont l'un des ajouts les plus importants pour le FPGA. Beaucoup de conceptions nécessitent l'utilisation d'un processeur embarqué. Souvent, le choix d'un dispositif de FPGA avec un processeur embarqué (comme le Virtex de Xilinx 5) permet de simplifier grandement le processus de conception tout en réduisant l'utilisation des ressources et la consommation d'énergie. Le PowerPC IBM 405 et 440 processeurs sont des exemples de deux processeurs inclus dans le Virtex 4 et 5 de Xilinx. Ce sont des processeurs RISC classiques qui mettent en œuvre un jeu d'instructions PowerPC.

1.2.5.4 Le gestionnaire d'horloge numérique (DCM)

Un gestionnaire d'horloge numérique permet d'avoir des périodes d'horloge différentes qui sont générées à partir d'une horloge de référence unique. La plupart des systèmes disposent d'une horloge externe unique qui produit une fréquence d'horloge fixe. Cependant, il y a un certain nombre de raisons pour lesquelles un concepteur peut avoir besoin de fonction logique fonctionnant à des fréquences différentes. L'avantage d'utiliser un DCM est que les horloges générées auront moins de gigue.

1.2.6 Configuration d'un FPGA

Pour configurer un FPGA, il faut un bitstream de configuration. C'est un fichier binaire dans lequel les informations de configuration pour un périphérique particulier sont stockées. Les bitstreams peuvent être partiels ou complets. Un bitstream complet (Merged bitstream) configure la mémoire de configuration de l'ensemble du dispositif. Il est utilisé pour la conception statique ou au début de l'exécution d'un système de reconfiguration dynamique afin de définir l'état initial des cellules SRAM. Le bitstream partiel configure une partie seulement du dispositif il est utilisé pour la reconfiguration partielle. Un FPGA fournit différents moyens interfaces de configuration, parmi eux il y a le port JTAG, l'interface SelectMAP pour la configuration de plusieurs FPGA, ou encore le port de configuration interne (ICAP) utilisé pour la reconfiguration.

1.3 La reconfiguration des FPGA

1.3.1 Introduction

Quelques années auparavant, les algorithmes qui pouvaient être mises en œuvre sur une puce FPGA unique été très peu répandue. En 1995, par exemple, les plus grands FPGA qui pouvaient être programmés disposaient d'environ 15 000 portes logiques au maximum. Plus récemment, les FPGA ont atteint le million de portes logiques. Ensuite, les chercheurs et les concepteurs ont vu la possibilité de réaliser des systèmes autoadaptatifs grâce aux FPGA. Pour cela le circuit logique doit être capable de procéder à une modification de sa fonctionnalité en cours de fonctionnement. Les FPGA basiques ne le permettent pas, il est nécessaire d'arrêter le système pour y implémenter une nouvelle configuration. Cela a amené les sociétés ATMEL et Xilinx à concevoir des architectures capables de procéder à une modification de leurs ressources sans arrêter le système. Cela donne naissance aux premières architectures reconfigurables dynamiquement. La capacité de l'architecture à s'adapter de manière dynamique à son environnement est caractérisée par le temps que met le système à se reconfigurer (quelques millisecondes à quelques nanosecondes) [Liu08]. Cette caractéristique est directement liée au temps d'accès de la mémoire SRAM de configuration. Le temps de reconfiguration dépend des technologies FPGA et des tâches matérielles utilisées, il peut varier de quelques microsecondes à quelques secondes. Les systèmes qui utilisent les architectures reconfigurables sont souvent utilisés pour réaliser des accélérateurs matériels pour traiter des tâches gourmandes en ressources et en temps de calcul [Ber92, VBR+96, FPPR04].

La mise en œuvre de ce type de fonctionnement permet de reconfigurer autant de fois que nécessaire l'architecture matérielle du système tout au long de son exécution. Les tâches matérielles sont alors obtenues par un découpage de l'application sous forme de partitions exécutées séquentiellement [PB99, Ama06]. Les ressources sur lesquelles sont exécutées ces partitions sont le résultat du découpage spatial du circuit reconfigurable. La manière dont un algorithme sera implémenté sur une architecture reconfigurable dynamiquement va déterminer d'une part le niveau de performance dans le traitement des données et d'autre part le niveau de dynamicité dont cette architecture pourra faire preuve.

1.3.2 Les différents modes de reconfiguration

1.3.2.1 Reconfiguration statique

Le FPGA peut modifier ses fonctionnalités matérielles, pour cela il doit arrêter son exécution, une nouvelle configuration matérielle est chargée sur le FPGA (il est aussi possible de charger un nouveau programme logiciel sur le processeur) et il faut redémarrer le système. Dans ce cas, la reconfiguration est un procédé indépendant de l'exécution de l'application. Ce procédé est appelé la reconfiguration statique, le contenu de la configuration reste statique pendant toute la durée de vie de l'application. Les figures 1.7 et 1.8 illustrent le cycle de fonctionnement et de reconfiguration pour une architecture reconfigurable statiquement, il est à remarquer que les demandes d'arrêt et de reprise des opérations sont réalisées à la demande de l'utilisateur. La reconfiguration statique est similaire à l'utilisation d'un ASIC d'un point de vue de l'application finale et de son utilisation. La reconfiguration statique n'est pas capable de répondre aux besoins dans le domaine des systèmes adaptatifs. Concrètement, la reconfiguration statique est utilisée pour les tests, pour le prototypage, pour les systèmes avec application figée et pour l'initialisation d'un système. Parmi les systèmes utilisant la reconfiguration statique, nous trouvons principalement les accélérateurs matériels dont le rôle est de réaliser des tâches gourmandes en temps de calcul et qui nécessitent une exécution rapide qui ne peut pas être réalisée sur des systèmes à base de processeur.

1.3.2.2 Reconfiguration dynamique

Une approche différente à la première est celle qui considère la reconfiguration du FPGA dans le cadre de l'application elle-même, en lui donnant la capacité d'adapter son matériel en configurant les ressources selon les besoins d'une situation particulière au cours de la durée d'exécution (figure 1.9). Ce cas particulier est appelé la reconfiguration dynamique. Dans

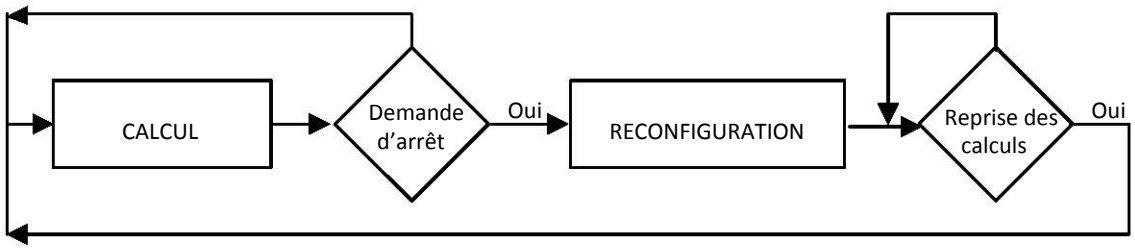


FIGURE 1.7 – Mécanisme de la reconfiguration statique d'un FPGA.

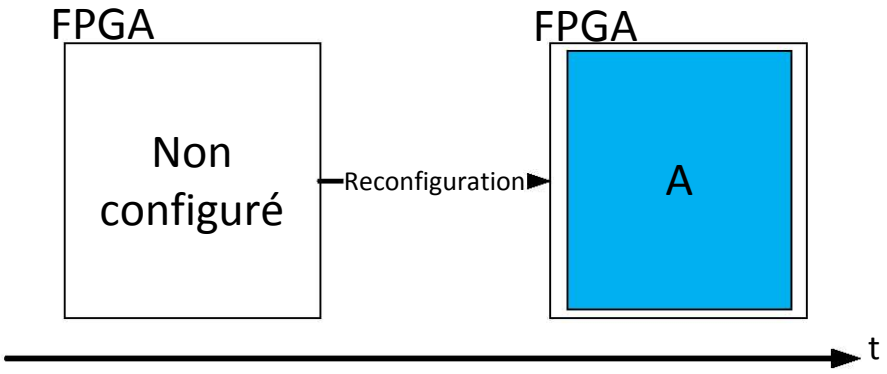


FIGURE 1.8 – Reconfiguration statique d'un FPGA.

le cas de la reconfiguration dynamique, le processus de reconfiguration est considéré comme faisant partie de l'exécution de l'application. La reconfiguration dynamique est caractérisée par une application qui a un nombre de configurations au moins égales à deux. La mise en œuvre d'une telle application se traduit par une première configuration du système suivie par des reconfigurations qui permettront de modifier l'application autant de fois que nécessaire. Le système se comporte comme un processeur dont les ressources matérielles s'adaptent au cours du temps. La reconfiguration dynamique permet principalement d'accélérer des tâches pour lesquelles un processeur ne serait pas adapté. Un système reconfigurable dynamiquement doit prendre en compte les temps de reconfiguration de celui-ci [Ama06]. Notamment pour les applications temps réel (multimédia, domotique, pilotage…) où les temps perdus pendant les phases de reconfigurations ne doivent pas impacter sur le temps de traitement du système [MRDW11a, DPW99, HSB06b].

Un des intérêts de la reconfiguration dynamique est la possibilité de modifier l'architecture matérielle du FPGA à l'infini pendant l'exécution d'une application comme le montre la figure 1.9. De plus, la reconfiguration dynamique est un moyen d'améliorer le principal handicap

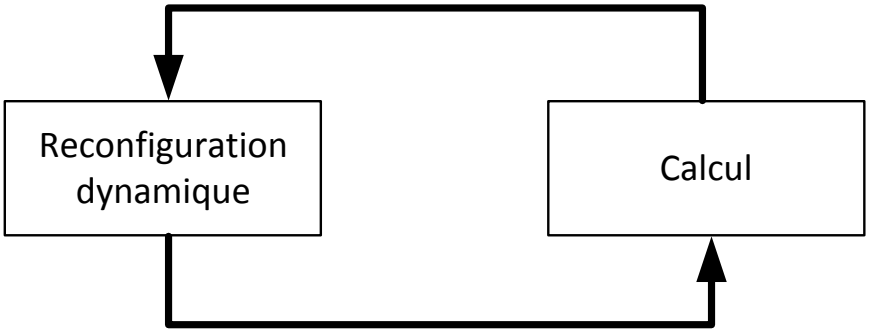


FIGURE 1.9 – Mécanisme de la reconfiguration dynamique d'un FPGA.

des FPGA par rapport aux ASICs. Le principal handicap des FPGA est qu'ils sont beaucoup moins denses en nombres de portes logiques que les ASIC. Grâce à la reconfiguration dynamique, une petite surface logique peut supporter une grande application ou plusieurs petites applications. La reconfiguration dynamique peut être divisée en deux modes de fonctionnement : la reconfiguration dynamique globale ou la reconfiguration dynamique partielle.

1.3.2.3 Reconfiguration dynamique globale

La reconfiguration dynamique globale permet de modifier la totalité du contenu de la mémoire de configuration. Dans ce type de reconfiguration, le FPGA est entièrement modifié pour accueillir une nouvelle implémentation [HB06]. La figure 1.10 illustre ce mode de fonctionnement qui consiste à reconfigurer la totalité du FPGA en passant d'une application à une autre. Sur cette figure, le FPGA est configuré avec une première architecture « A », ensuite il est entièrement reconfiguré pour passer à l'architecture « B ». La reconfiguration dynamique globale est une évolution de la reconfiguration statique. Cette évolution permet au FPGA de supporter une plus grande flexibilité et une bonne adaptation à son environnement. Actuellement avec les évolutions technologiques des FPGA nous pouvons implémenter de plus en plus d'applications dans un même circuit. Par conséquent, le problème de la taille des FPGA est de moins en moins contraignant, mais il reste toujours présent, car le coût en ressources logiques d'un FPGA est important.

1.3.2.4 Reconfiguration dynamique partielle

Le principal inconvénient de la reconfiguration dynamique globale est l'obligation de modifier la totalité du circuit même si certaines parties sont communes, comme le montre l'exemple

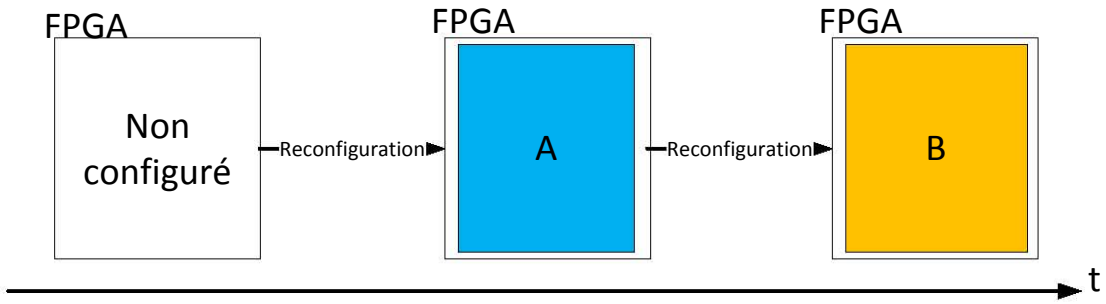


FIGURE 1.10 – Reconfiguration dynamique globale d'un FPGA.

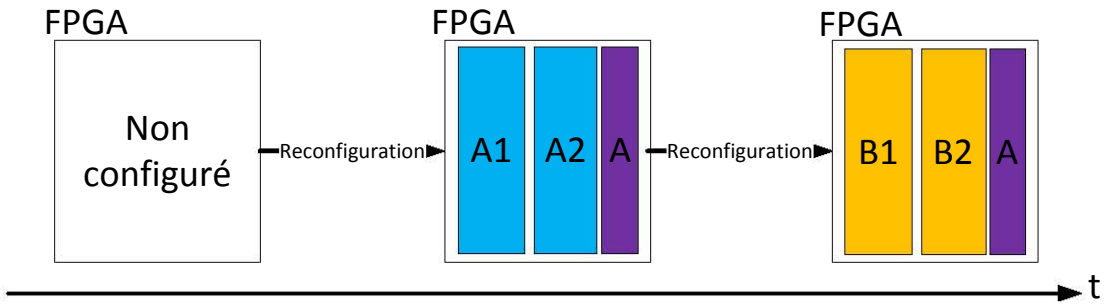


FIGURE 1.11 – Reconfiguration dynamique partielle d'un FPGA.

de la figure 1.11. Pour résoudre cet inconvénient, il existe une solution appelée la reconfiguration dynamique partielle. Elle permet de modifier une ou plusieurs parties du circuit sans interrompre le fonctionnement de celui-ci. La figure 1.11 illustre ce mode de fonctionnement qui consiste à reconfigurer certaines régions. Dans un premier temps, le FPGA est configuré avec une première configuration « A, A1, A2 ». Après une reconfiguration, la nouvelle configuration est « A, B1, B2 ». « A1 et A2 » ont respectivement été remplacés par « B1 et B2 » tandis que « A » n'est pas modifiée et est toujours en cours d'exécution.

1.4 Conclusion

Ce chapitre nous a permis de montrer que la reconfiguration dynamique permet d'augmenter l'efficacité d'un FPGA en allouant des ressources logiques à plusieurs tâches. Ainsi, en utilisant la mémoire externe pour stocker les fichiers de reconfigurations, il est possible d'implémenter une application de taille plus grande que la mémoire physique disponible sur le FPGA. Par conséquent, un FPGA reconfigurable dynamiquement peut offrir une quantité plus importante de ressources logiques qu'il en dispose.

Un autre avantage qui suscite un intérêt croissant est la flexibilité introduite par la reconfiguration dynamique des FPGA qui se manifeste à deux niveaux. Le premier niveau permet au concepteur, d'adapter facilement son application pour faire face à de nouvelles contraintes ou pour remplacer un algorithme par un autre plus efficace. Cette flexibilité assure une certaine évolution du système. Le second niveau se situe sur le choix des tâches à exécuter, qui peut se faire dynamiquement en fonction des données à traiter. On peut donc tout à fait imaginer que les données d'entrée puissent influer en temps réel sur l'enchaînement des algorithmes. Le matériel devient extrêmement malléable et contrôlable par le logiciel.

Dans la suite de ce travail de thèse, nous allons présenter en détail les techniques utilisées pour la reconfiguration dynamique.

Chapitre 2

Reconfiguration partielle des FPGA

Sommaire

2.1 Introduction

La reconfiguration est un terme composé du préfixe « re » qui exprime la réitération (refaire), le retour à un état antérieur (recourber), le renforcement (repenser), etc. Donc la reconfiguration est une réitération d'une configuration, par conséquent, c'est une nouvelle modification ou un nouveau réglage de paramètres qui permet l'optimisation du fonctionnement d'un système. Pour les FPGA la reconfiguration désigne le fait que l'architecture matérielle du système soit capable de s'adapter en changeant de forme ou de fonction. Le FPGA reconfigurable peut donc changer sa configuration à n'importe quel moment. Cette reconfiguration peut être dynamique, car le processus de réitération permet d'avoir à chaque nouvelle configuration une nouvelle action. On peut donc définir les FPGA comme des systèmes reconfigurables et c'est leur exécution qui est dynamique. Tout système permettant une modification de son architecture grâce à l'anticipation et/ou à la réaction est auto-adaptatif. Et toute architecture matérielle permettant une modification de forme ou de fonction à tout moment est adaptable dynamiquement. Par conséquent, un système auto-adaptatif à base de FPGA utilise la reconfiguration dynamique pour s'auto-adapter à son environnement. Pour réaliser cette adaptation et avoir un système flexible, il faut utiliser des techniques de conception et des architectures adéquates. Toutes ces architectures et ces techniques de conception sont en constante évolution pour améliorer la flexibilité des FPGA reconfigurables.

La suite de ce chapitre est organisée de la manière suivante : dans une première partie, nous définirons la terminologie pour la reconfiguration utilisée dans la suite de ce manuscrit de thèse. Ensuite, une présentation des techniques de reconfiguration des FPGA. Puis, nous verrons les travaux et solutions proposées dans les différents travaux de recherche. Pour finir, nous présenterons les limitations et problèmes rencontrés pour le placement des tâches matérielles de tailles variables.

2.2 Terminologies pour la reconfiguration

La terminologie spécifique au domaine de la reconfiguration est précisée ci-dessous et dans la figure 2.1).

– Région reconfigurable : Région délimitée initialement dans le circuit FPGA et destinée à être le siège des reconfigurations dynamiques.

– Région statique : Région délimitée qui, à l'inverse de la région reconfigurable, est destinée à un fonctionnement purement statique.
– Partition Reconfigurable Partiellement (Partial Reconfigurable Partition) (Partition PR) : est une partition de la région reconfigurable. Une région reconfigurable peut être composée de plusieurs partitions PR.
– Tâches matérielles ou Intellectual Property (IP core) : Blocs fonctionnels complexes réutilisables.
– Module Partialement Reconfigurable (Module PR) : tâche matérielle qui peut peupler une région reconfigurable. Il peut y avoir plusieurs modules reconfigurables par région. Un module PR peut aussi occuper une ou plusieurs partitions PR.
– Wrapper : Circuit matériel permettant de connecter une tâche matérielle au reste du système reconfigurable.
– Interfaces de connexions : Point de connexion d'une région reconfigurable.

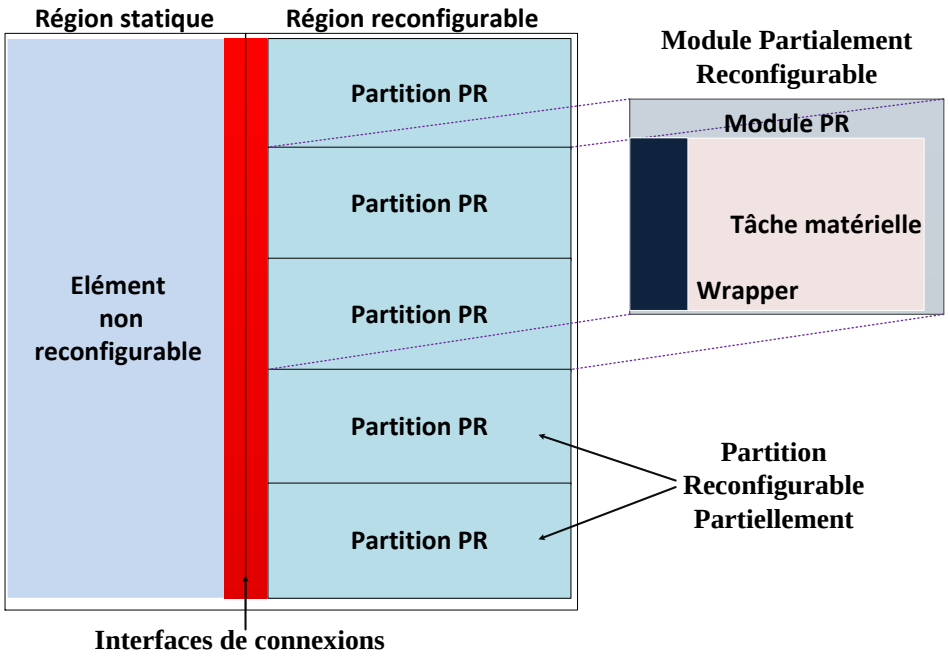


FIGURE 2.1 – Terminologies pour la reconfiguration.

2.3 Techniques de reconfiguration des FPGA

La reconfiguration existe depuis plusieurs années, son évolution a permis de dégager différentes méthodes. Nous allons voir dans un premier temps les moyens dont nous disposons pour utiliser la reconfiguration. Ensuite nous verrons qu'il existe plusieurs flots de conception pour mettre en œuvre la reconfiguration partielle des circuits FPGA de chez Xilinx. Puis nous établirons un comparatif sur les systèmes d'interconnexions des régions reconfigurables. Pour finir, nous verrons les différentes contraintes de placement en fonction des technologies et des flots utilisés.

2.3.1 Moyens pour la reconfiguration partielle

La reconfiguration d'un FPGA peut être réalisée sur le FPGA (en interne). Dans le cas d'une demande de reconfiguration externe au FPGA, un PC dialogue avec le FPGA. Cette communication est réalisée par l'intermédiaire du port JTAG du FPGA. Les bitstreams de reconfigurations partielles ou complets sont envoyés sur le FPGA. Cette méthode demande l'action d'un acteur extérieur et elle n'est donc pas adéquate pour un système autoadaptatif.

La reconfiguration d'un système autoadaptatif sur les FPGA Xilinx est mise en œuvre en se servant du "Port d'Accès de Configuration Interne" (ICAP). Un bitstream partiel est écrit dans l'ICAP, ce bitstream reconfigure alors les zones spécifiées du FPGA. La communication avec l'ICAP peut être mise en œuvre par un microprocesseur embarqué (un PowerPC ou un Microblaze).

L'ICAP fournit un accès interne à la logique de configuration du FPGA. Grâce à l'interface de l'ICAP, les données de configuration peuvent être chargées dynamiquement dans la mémoire de configuration du FPGA. Il est également possible d'effectuer une relecture des données de configuration de la mémoire de configuration ou de lire les registres d'état de la logique de configuration avec l'interface ICAP. Comme le montre la figure 2.2, les interfaces de l'ICAP vers la mémoire de configuration lui permettent d'accéder aux ressources configurables du FPGA. L'interface ICAP comprend des ports de données distinctes pour la lecture (O) et l'écriture (I) des données de configuration. La largeur des ports de données de configuration peut être configuré pour être de 8 ou 32 bits pour les Virtex-4, et de 8, 16 ou 32 bits pour les Virtex-5 et 6 [Xilf, Xile].

Xilinx précise que l'interface ICAP ne doit pas être cadencée plus rapidement que la fré-

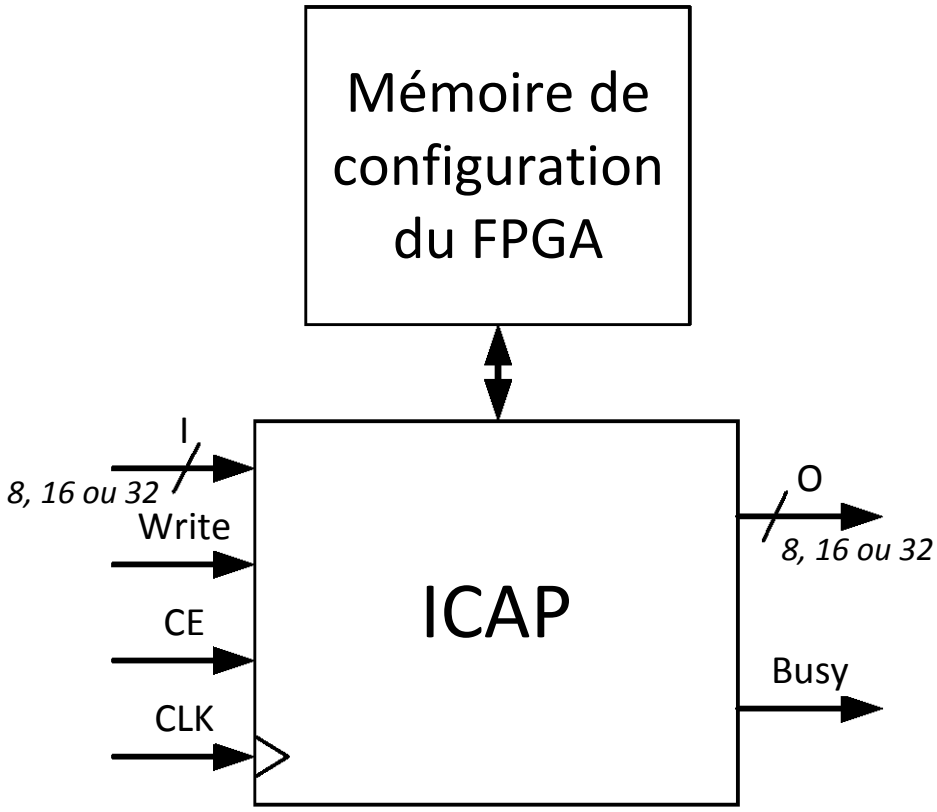


FIGURE 2.2 – Architecture de l'ICAP (Port d'Accès de Configuration Interne).

quence recommandée par défaut de 100 MHz. Avec une fréquence de 100 MHz et une largeur de 32 bits d'interface de données, l'interface ICAP fournira alors une vitesse maximale de reconfiguration par défaut de 3200 Mbit/s ou 400 Mo/s (si le bitstream est stocké sur les BRAM du FPGA).

2.3.2 L'évolution des flots de conception

Dans cette parti nous allons décrire les flots de conception pour la reconfiguration partielle du plus ancien au plus récent.

2.3.2.1 Conception modulaire de la reconfiguration partielle

La technique la plus ancienne est appelée « Module-based partial reconfiguration » (MBPR) elle permet la reconfiguration de modules prédéfinies au préalable [Xilb]. Le but de cette méthode est de reconfigurer des portions distinctes d'un FPGA tandis que le reste du dispositif

continue de fonctionner. Cette méthode est basée sur les flots de conception modulaire. La conception modulaire est une méthode de développement conçue par Xilinx [XLP]. Cette méthode permet à une équipe d'ingénieurs de travailler de façon autonome sur différents modules d'une conception et de les fusionner ensuite en une seule conception pour un FPGA. Chaque module peut être conçu par un membre de l'équipe tandis que le chef d'équipe travaille toujours sur la conception de haut niveau. Cette méthode permet un gain de temps et permet de modifier un module tout en laissant les autres inchangés, parce que chaque module est considéré comme un module indépendant. Par contre, la communication entre les modules et le reste de la conception doit être assurée par des bus macro placés aux frontières des modules. Le bus macro fournit une communication fixe entre les différents modules de la conception. Tous les modules de base qui seront partiellement reconfigurables devront contenir un ensemble de connexions compatible avec les bus macro. Les connexions sont réalisées et placées à la main pendant la phase de conception. Sans cette attention particulière, les communications avec les modules reconfigurables ne seraient pas possibles, car il serait impossible de garantir l'acheminement des communications entre les modules. Pour chaque reconfiguration partielle effectuée, le bus macro est utilisé pour établir un routage invariable avec les autres modules statiques ou reconfigurables, en garantissant des connexions correctes.

Les bus macro [LBM+06] permettent la communication à travers des buffer à trois états. Il faut donc veiller à ce que des précautions soient prises pour que les buffers à trois états ne soient pas en cours de modification pendant une reconfiguration. Les bus macro [LBM+06] sont définis par une description HDL. Cette description doit faire en sorte que tous les signaux du module reconfigurable qui sont utilisés pour communiquer avec un autre module utilisent un bus macro comme le montre la figure 2.3. L'insertion des bus macro doit être réalisée de telle sorte qu'ils ne chevauchent pas une colonne de type DSP ou type BRAM. Les bus macro sont le moyen de communication entre les modules reconfigurables et la partie statique, par conséquent toutes les connexions passent par un bus macro à l'exception des signaux d'horloge.

L'architecture d'un bus macro montrée sur la figure 2.4 montre qu'un bus macro offre 4 bits de communication. La mise en œuvre actuelle d'un bus macro utilise huit buffers à trois états (TBUFs) reliés par plusieurs lignes qui permettent à un bit d'information de passer soit de gauche à droite ou de droite à gauche. Un bus macro utilise deux bus d'entrées (Ai et Bi) et deux bus de sorties (Ao et Bo) chacun sur quatre bits. Le bus macro utilise aussi deux bus de configuration (As et Bs) sur quatre bits pour modifier l'état des buffers à trois états. Par exemple,

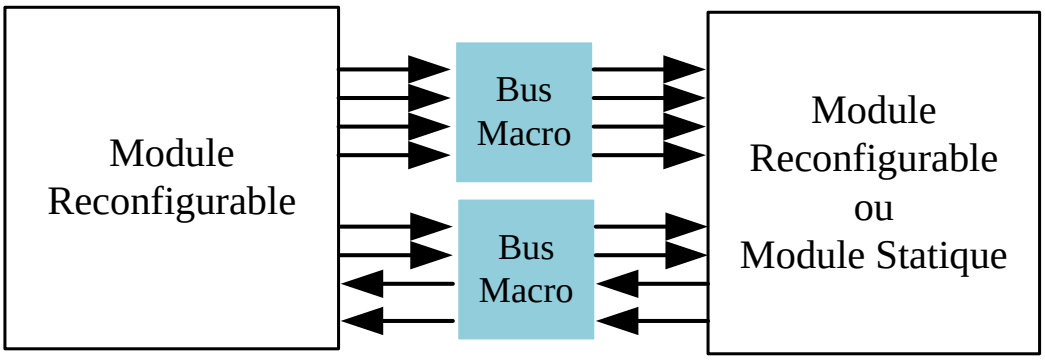


FIGURE 2.3 – Placement des bus macro.

si on souhaite que le bus macro établisse une commutation sur 1 bit entre le module de droite et celui gauche, on doit définir As(0) à 1 et Bs(0) à 0. Ensuite, on connecte le module émetteur sur Ai(0) et le module récepteur sur Bo(0).

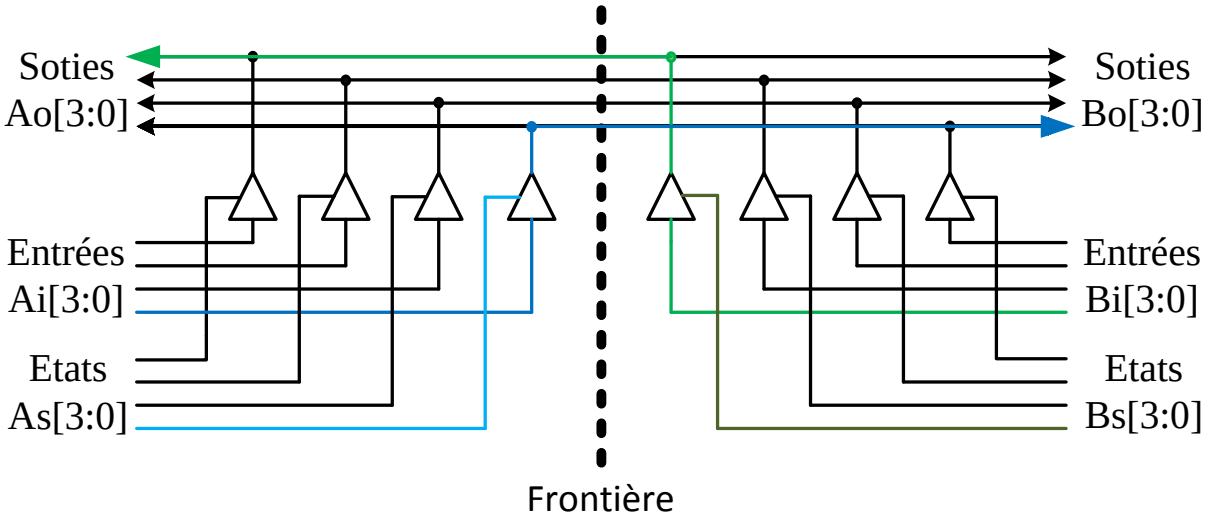


FIGURE 2.4 – Architecture d'un Bus Macro.

La phase d'assemblage finale est le processus consistant à combiner chacun des modules individuels dans une conception FPGA complète. Le placement et le routage sont réalisés au cours de la phase d'implémentation de chaque module. Le placement et le routage sera ensuite conservée pour maintenir les connections et les performances de chaque module. Chaque module reconfigurable sera placé sur une colonne reconfigurable de l'architecture du FPGA. Le flot de reconfiguration partielle exige que le bitstream initial qui est chargé dans le dispositif FPGA soit une conception complète. Ceci est nécessaire afin que tous les modules non reconfi-

gurables soient placés et verrouillés, et que seuls les éléments reconfigurables de la conception vont changer lors de la reconfiguration partielle.

2.3.2.2 Difference-based partial reconfiguration

Par la suite une nouvelle technique a vu le jour appelé « Difference-based partial reconfiguration » (DBPR) [XE]. Elle peut-être utilisée quand un petit changement est apporté à la conception. Elle est particulièrement utile en cas de changement d'équation des LUTs ou d'une mémoire dédiée. Le bitstream partiel ne contient que des informations sur les différences entre la structure de conception actuelle (qui réside dans le FPGA) et le nouveau contenu du FPGA. Il y a deux méthodes de différence basée sur la reconfiguration connue (front-end et back-end). La première est basée sur la modification de la conception dans les langages de description matérielle (HDL). Il est clair qu'une telle solution exige la pleine répétition de la synthèse et des processus de mise en œuvre. Le back-end permet d'apporter des modifications au stade de la mise en œuvre du prototypage. Par conséquent, il n'est pas nécessaire de resynthétiser la conception. L'utilisation de ces deux méthodes aboutit à la création d'un bitstream partiel grâce à la comparaison de deux conceptions qui peut être utilisée pour une reconfiguration partielle du FPGA. La commutation de la configuration d'un module à un autre est très rapide, vu que les différents bitstream peuvent être plus petites. Cette méthode ne peut pas être entièrement adaptable, car les modifications apportées à la conception ne sont pas importantes et l'intégrité du signal n'est pas assurée sur les limites des modules reconfigurables. C'est pourquoi nous ne couvrirons pas cette méthode en détail.

2.3.2.3 Early Access Partial Reconfiguration

Un flot d'accès rapide à la reconfiguration partielle a vue le jour pour pallier aux contraints des deux flots précèdents. Ce flot nommé Early Access Partial Reconfiguration (EAPR) [Xild] permet de faciliter la conception d'un système reconfigurable partiellement. Il est possible avec ce flot d'adapter une région à un module partiellement reconfigurable. L'exigence du flot MBPR de placer l'ensemble des régions reconfigurables sur des colonnes est supprimée. Le flot EAPR permet désormais aux régions reconfigurables d'être de toute taille tant que sa forme est rectangulaire. La plus petite région reconfigurable qui peut être définie est une «frame», qui s'étend sur toute la hauteur d'un domaine d'horloge du FPGA.

De plus, un nouveau système de communication appelé slice bus macro a été développé par

Hübner et. al. [HBB04] comme alternatives aux bus macro (basée sur des buffers trois états). Les bus macro basés sur des slices (ou des LUT) sont représentés sur la figure 2.5, chaque bus macro se compose de deux blocs logiques configurables (CLB). Les bus macro basés sur les slices comportent un certain nombre d'avantages par rapport à ceux basés sur des buffers trois états. Le premier avantage est dû à l'évolution de la technologie des slices bus macro utilisent des LUT ceci est très important vu que les buffers à trois états ont disparu depuis la série 4 pour les FPGA Virtex. Ils permettent aussi d'améliorer considérablement les performances de synchronisation et simplifient le processus de construction d'une conception. De plus des fonctionnalités supplémentaires ont été ajoutées pour désactiver les signaux sortants. Cette fonction est très utile, car le comportement des signaux émanant du module n'est pas connu pendant une reconfiguration (ou si aucun module n'est présent). Comme on peut le voir sur la figure 2.6 les slice bus macro sont placés dans les modules alors que le bus macro se trouves entre les modules.

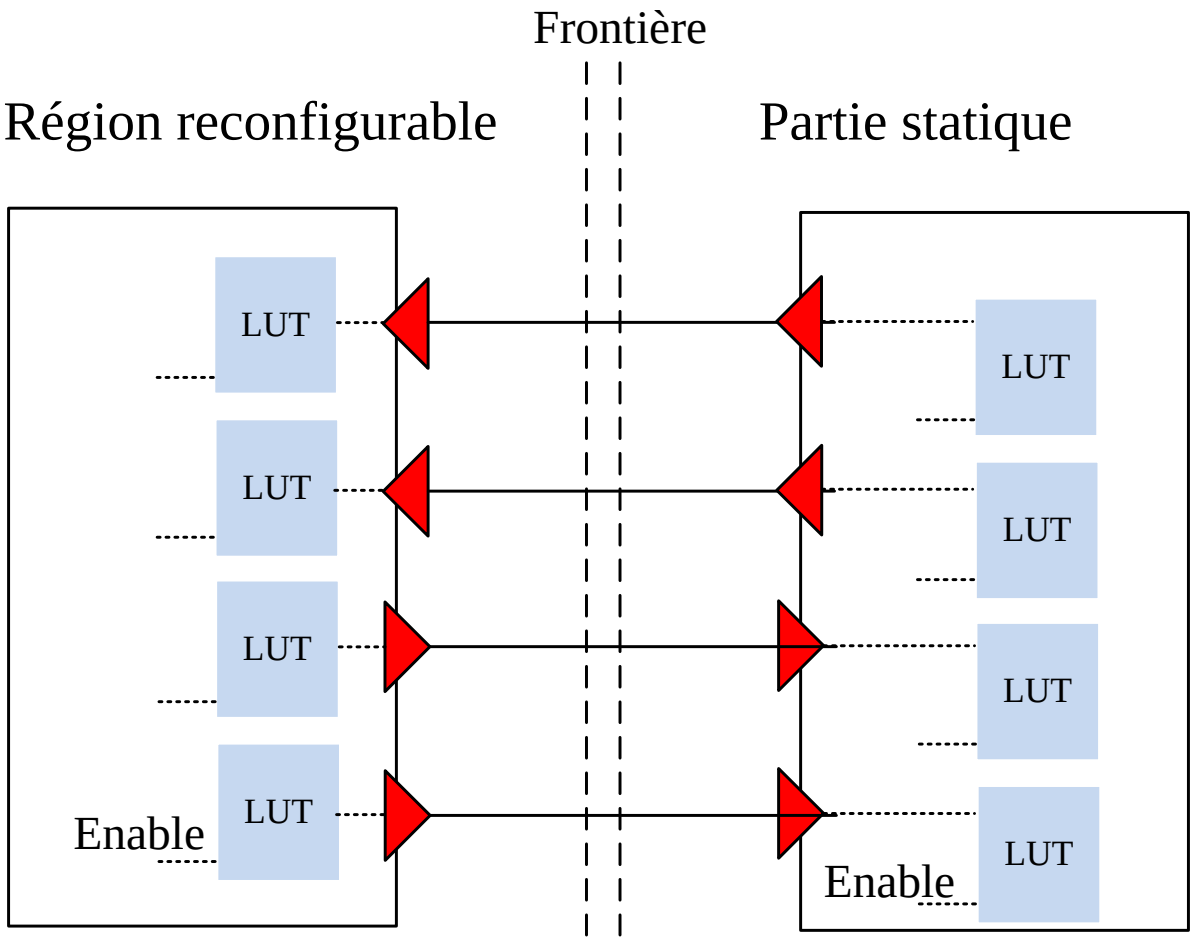


FIGURE 2.5 – Architecture d'un Slice bus macro.

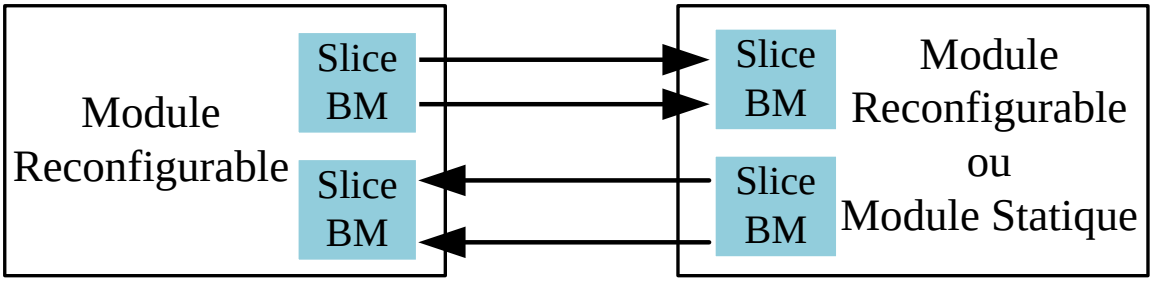


FIGURE 2.6 – Placement d'un Slice bus macro.

2.3.2.4 Partition based partial reconfiguration

Divers inconvénients du flot EAPR ont été résolus avec l'apparition du flot « Partition based partial reconfiguration » (PBPR). Premièrement, ce flot permet un plus grand degré de liberté sur la forme des régions reconfigurables par rapport aux flots précédents. Cette liberté permet de sélectionner au mieux les ressources nécessaires. Prenons comme exemple une IP qui a besoin d'une slice BRAM, d'une slice DPS et de vingt slices CLB. La figure 2.7 vous présente trois choix de ressources pour implémenter cette fonction. Le choix (a) correspond à une sélection des ressources rectangulaires (flot MBPR, DBPR et EAPR), cette sélection n'est pas optimale en terme de slice BRAM et DSP. La région reconfigurable occupe deux slices BRAM et deux slices DSP alors que la fonction à implémenter utilise une slice BRAM et une slice DSP. Grâce au flot PBPR nous pouvons améliorer cette sélection de ressources en modifiant la forme de la région reconfigurable. Les choix b et c permettent une occupation à 100 % de la région reconfigurable par la fonction implémentée. Les articles [MRDW11b, DNMR+12] nous ont permis de valider cette nouvelle fonctionnalité du flot PBPR. Dans ces articles, nous avons comparé l'ancien et le nouveau flot sur des IP d'encodage vidéo pour la haute définition et la basse définition. Grâce à une meilleure sélection des ressources, nous avons montré sur cet exemple que l'on pouvait atteindre un gain de 30 % sur l'utilisation des ressources des régions reconfigurables.

Deuxièmement, les bus macro ont été remplacés par des pins partitions. Ces pins partitions sont nécessaires pour garantir la communication entre les parties statiques et les régions reconfigurables. Les pins partitions sont différents des bus macro, leur architecture est équivalente à une LUT sur un bit. Par rapport à l'architecture des slices bus macro l'occupation sur le FPGA a été divisée par deux, on est passé de deux LUT a une seule. Un autre avantage est que les pins partition sont automatiquement placées sur le FPGA, il n'est plus nécessaire de le faire

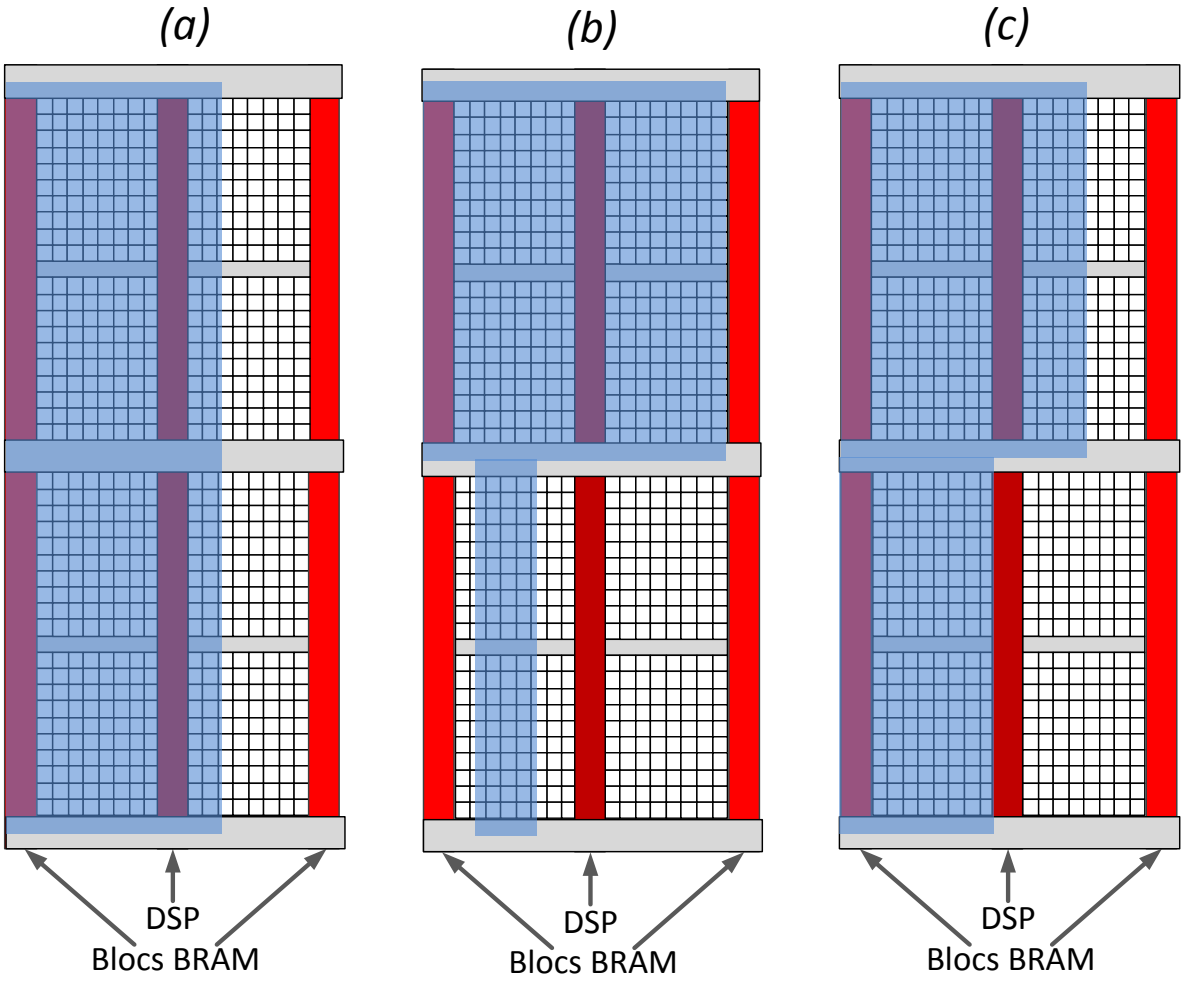


FIGURE 2.7 – Différentes formes pour les régions reconfigurables.

manuellement. Comme le montre les figures 2.9 2.8, elles sont placées dans la région reconfigurable sur les slices disponibles. Une slice peut contenir plusieurs pins partitions d'entrées ou de sorties. Prenons l'exemple d'un Virtex 5, une slice peut contenir quatre pins partitions (une pin partition par LUT disponible).

2.3.3 Comparatif sur les systèmes d'interconnexion des modules reconfigurables

Comme nous avons pu le voir précédemment les systèmes d'interconnexion pour les modules reconfigurables ont évolué en même temps que les flots de conception (Tableau 2.1). En 2002 les Bus Macro sont introduits, ils permettent de réaliser les premières communications avec les ports des modules reconfigurables. Huebner et coll. [HBB04] propose en 2004 une

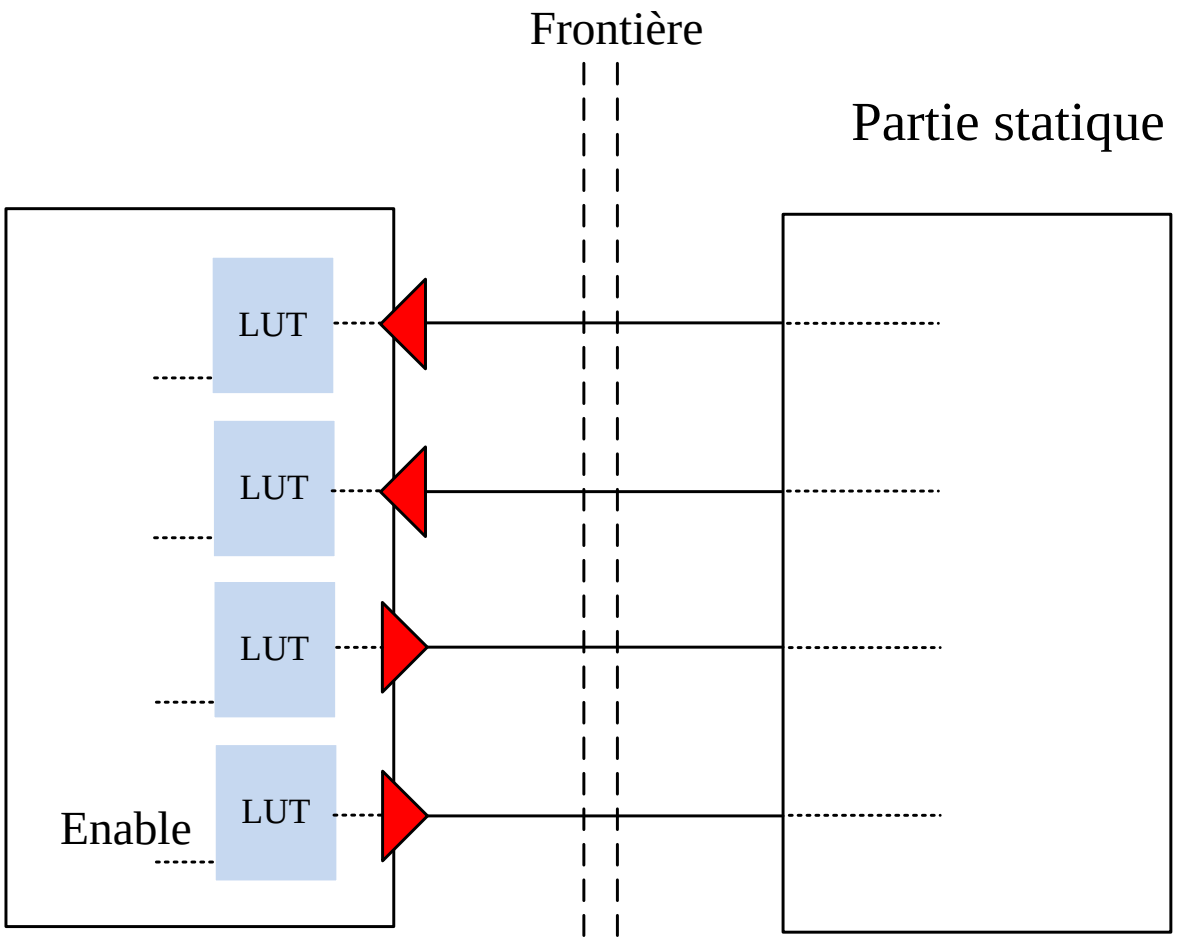


FIGURE 2.8 – Architecture d'un Pin Partition.

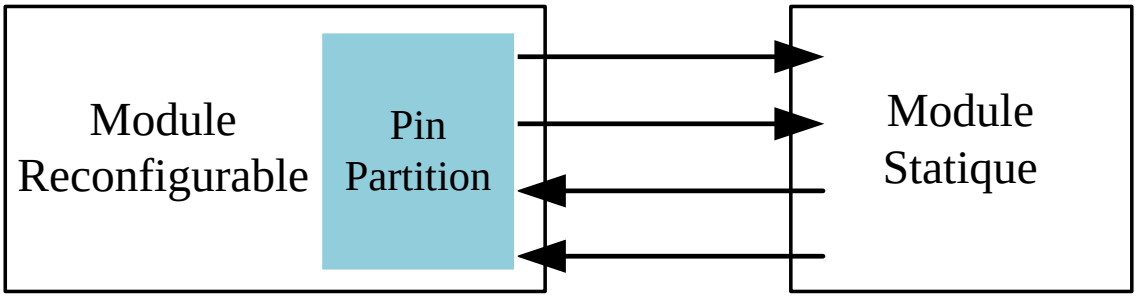


FIGURE 2.9 – Placement des pins partitions.

évolution qui utilise des LUT et non des buffers à trois états. Cette évolution va permettre de simplifier le flot de conception et l'implémentation des interconnexions des modules reconfigurables. En 2009 Xilinx propose les pin partition qui simplifient grandement le flot de conception grâce au placement automatique des interconnexions, de plus les pin partition sont moins cou-

teux en terme d'occupation sur le FPGA. Toutes ces interfaces de communication représentent

TABLE 2.1 – comparatif des interfaces de communication.

Type	Bus Macro	Slice Bus Macro	Pin Partition
Date de création	Introduit en 2002	Proposé en 2004 [HBB04]	Introduit par Xilinx en 2009
Architecture	2 buffer à 3 états par signal	2 LUT par signal	1 LUT par signal
Placement	Entre les modules	Dans les modules statiques et reconfigurables	Seulement dans les modules reconfigurables
Implémentation	Manuel et restreinte	Manuel	Automatique
Compatibilité FPGA	FPGA avec des buffers à trois états	Virtex 2-Pro, Virtex 4	Virtex 5 et 6
FLOT	MBPR / DBPR	EAPR	PBPR

la connexion entre le wrapper d'une IP statique et le wrapper d'une IP reconfigurable ou entre les deux wrappers de deux IP reconfigurables. Le wrapper correspond à l'interface de communication d'un IP core. Il est défini par les ports d'entrées et de sorties d'une IP. La région reconfigurable doit avoir un wrapper commun avec les IP qui seront placés sur celle-ci. Par conséquent, les wrappers des régions reconfigurables doivent être figés. Pour les différents flots présentés ci-dessus, les IP implémentées sur une même région reconfigurable doivent avoir un wrapper commun, cela signifie qu'elles auront le même nombre d'entrées et de sorties, des entrées et sorties de même taille et de même nom. La figure 2.10 montre un exemple de deux IP core (bleu et verte) implémentées sur la même région reconfigurable. On remarque que les wrappers des IP sont identiques à celui de la région reconfigurable. Chaque bit d'entrée ou de sortie est connecté à une interface de communication (Bus macro ou Slice Bus macro ou Pin partition) à part le bit d'horloge qui sera connecté à l'arbre d'horloge du FPGA.

2.3.4 Placement

Les différentes familles de FPGA ont des contraintes de configuration différentes pour le placement et l'implémentation de ces régions reconfigurables en fonctions des flots de conception compatible (PBPR et EAPR pour les FPGA Virtex 2/2 Pro, EAPR pour les FPGA Virtex 4 et PBPR pour les FPGA Virtex 5/6).

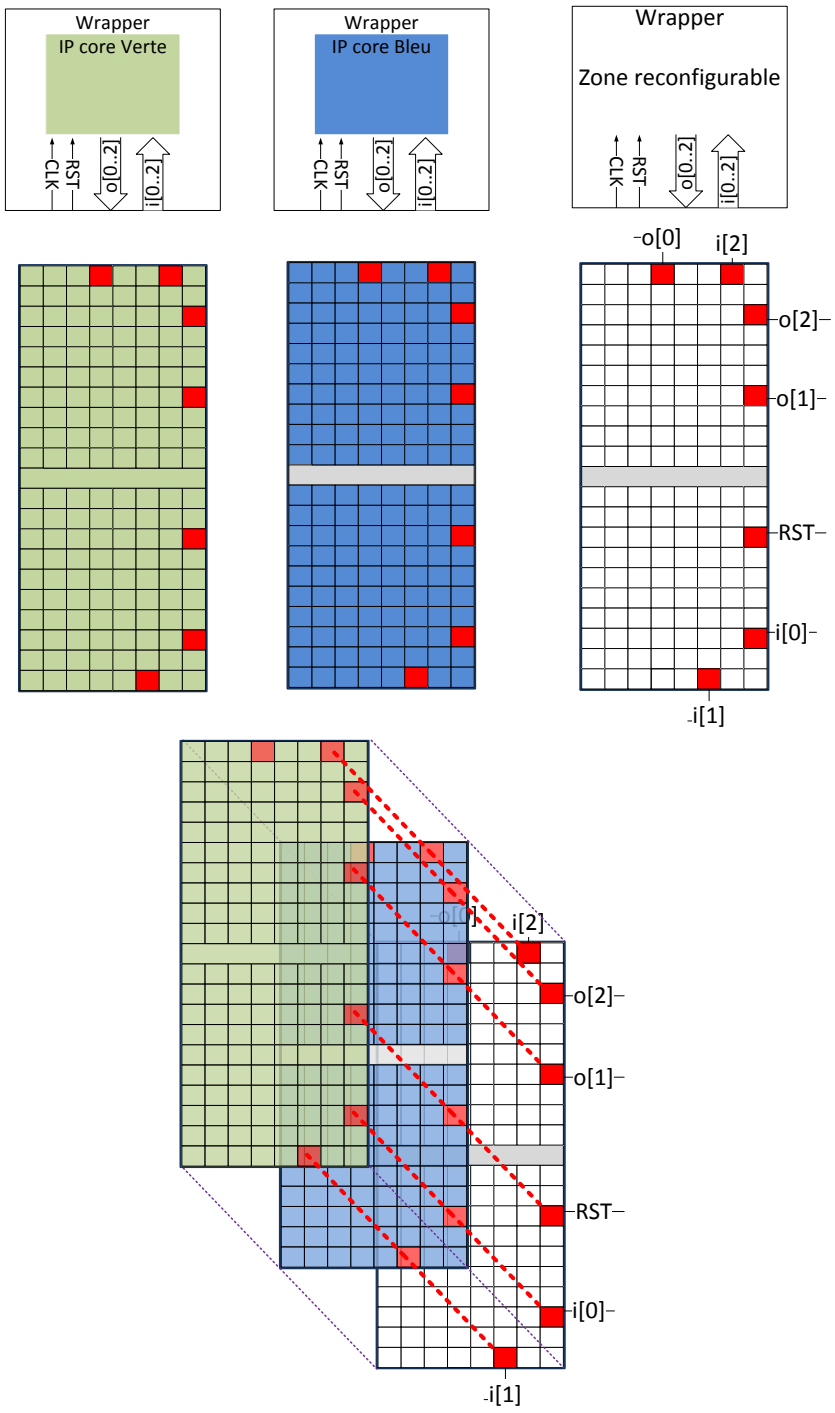


FIGURE 2.10 – Exemple d'implémentation d'IP sur une région reconfigurable.

Pour la famille de FPGA Virtex 2/2 Pro de chez Xilinx, l'unité basique de configuration est une colonne CLB qui s'étend sur l'ensemble du FPGA. Dans l'ancien flot de reconfiguration MBPR, la taille des régions reconfigurables doit être un multiple d'une colonne CLB complète.

Avec le nouveau flot EAPR la région reconfigurable n'est plus obligée de prendre une colonne complète, mais il n'est pas possible d'avoir deux régions reconfigurables différentes sur la même colonne. La région reconfigurable peut être définie sur une partie ou sur la totalité d'une colonne CLB, la partie restante peut être allouée pour l'utilisation d'une région statique. Comme on peut le voir sur la figure 2.11, PRR-1 et PRR-2 sont correctement placées pour le flot EAPR, par contre PRR-2 n'est pas compatible avec le flot MBPR car elle n'utilise pas une colonne complète. PRR-3/4 et PRR-5/6 se chevauchent sur la même colonne, leurs placements ne sont donc pas corrects pour les deux flots.

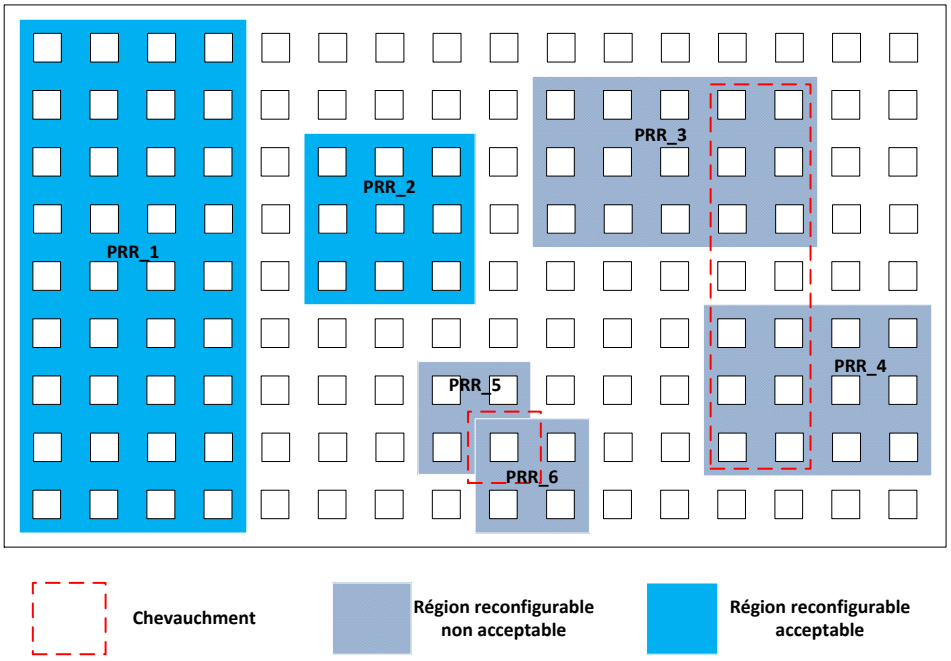


FIGURE 2.11 – Placement des régions reconfigurables sur un FPGA Virtex 2/2 Pro de chez Xilinx.

Pour les familles de FPGA Virtex 4/5/6 de chez Xilinx, l'unité basique de configuration est une colonne 16*1 CLB pour les Virtex 4, 20*1 CLB pour les Virtex 5 et 40*1 CLB pour les Virtex 6. La taille de ces colonnes correspond au différent domaine horloge du FPGA. De plus comme on peut le voir sur la figure 2.12 les domaines d'horloges sont divisés en deux par une ligne d'entrée sortie qui se trouve dans la partie centrale du FPGA (pour le Virtex 6, il peut y avoir deux lignes d'entrée sortie dans la partie centrale du FPGA). Une PRR ne peut pas chevaucher directement une ligne d'entrée sortie vu que les blocs d'entrée et de sortie interagissent avec les entrées et les sorties externe au FPGA et qu'elles ne sont pas reconfigurables.

Par conséquent, le placement de PRR-5 n'est pas correct. Comme pour les Virtex 2/2 Pro une unité basique reconfigurable ne peut être allouée que par une seule région reconfigurable donc PRR-8 et PRR-9 ont un placement incorrect. PRR-7 et PRR-6 se chevauchent, leurs placements ne sont donc pas corrects. Par contre PRR-1 et PRR-2 ont des placements tout à fait corrects. Pour PRR-3 son placement est correct si l'on utilise le flot de conception PBPR mais il ne sera pas possible de réaliser cette forme de région reconfigurable avec le flot EAPR qui oblige à avoir des formes rectangulaires. Une fois le placement réalisé correctement en fonction du FPGA et du flot de conception utilisé on doit modifier la taille des régions reconfigurables en fonction des modules que l'on souhaite implémenter dans la PRR. La taille PR devra être équivalente au module qui nécessite l'allocation du plus grand nombre de ressources. Une fois choisie la taille de la PR ne pourra plus être modifiée.

2.4 Contraintes de mise en œuvre de la reconfiguration

2.4.1 Éléments de communication

La communication entre les différents modules d'un système est un aspect fondamental de celui-ci. Cette communication permet le bon acheminement des données. Avec l'apparition de la reconfiguration, la communication doit être flexible pour s'adapter au différentes tâches matérielles. Plusieurs types de communications entre modules sont possibles sur une architecture (point à point, BUS, crossbar, réseau sur puce).

La communication point à point (figure 2.13.a) est la plus facile à mettre en œuvre. Les modules sont directement reliés entre eux par l'intermédiaire de ports dédiés à l'échange de données. Ce type de connexion est sûr de fonctionnement et très rapide, par contre une fois réalisée elle est figée et impossible à modifier. Cette structure limite toute flexibilité du système

La communication par bus (figure 2.13.b) permet une communication entre les modules [JS00]. Un bus de communication peut être partagé par plusieurs modules (software ou hardware). On peut distinguer trois types de bus partagés (backplane, Processeur-Mémoire et Entrée-Sortie). Le bus backplane permet la communication entre les processeurs, les mémoires et les blocs entrée/sortie. Le bus Processeur-Mémoire dispose d'une grande bande passante, il est utilisé dans la communication entre les processeurs et les mémoires. Pour finir, le bus d'entrée-sortie est caractérisé par une bande passante plus faible. Chacun de ces bus peut être décomposé en trois sous bus. Un bus unidirectionnel d'adresses qui transporte les adresses mémoires permettant

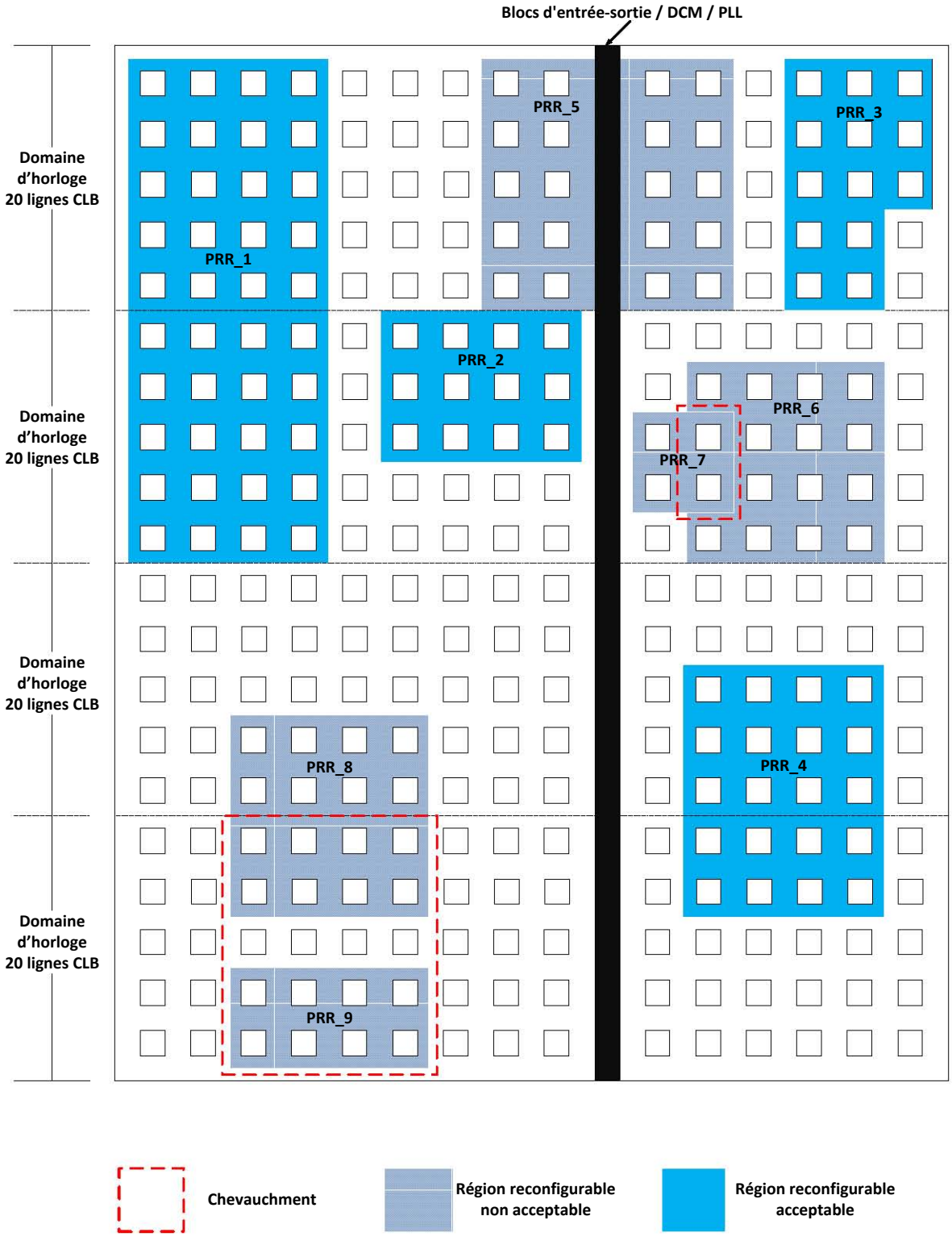


FIGURE 2.12 – Placement des régions reconfigurables sur un FPGA Virtex 5 de chez Xilinx (Placement identique pour les Virtex 4 et 6.

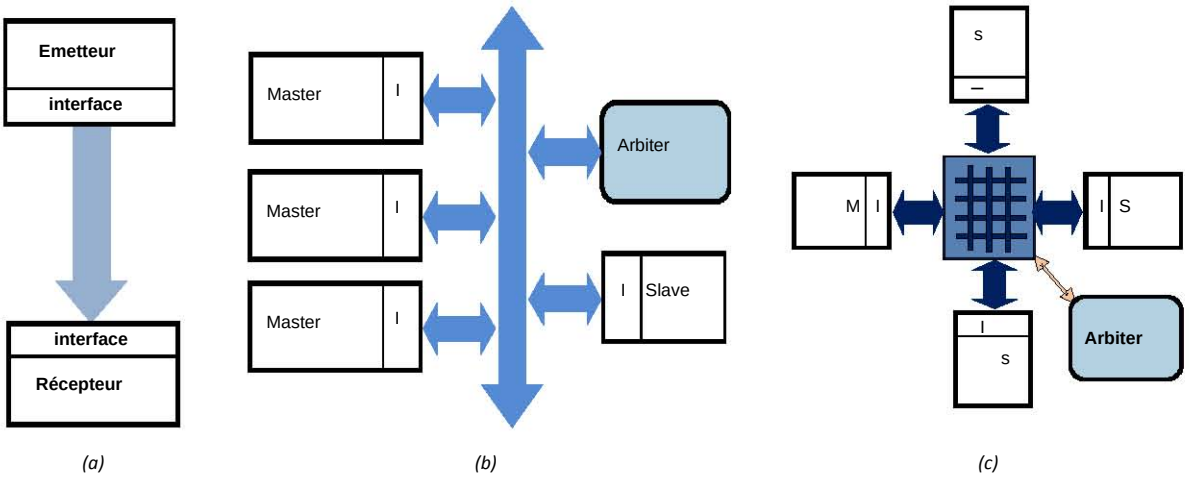


FIGURE 2.13 – Communication (a) point à point, (b) bus ,(c) réseau crossbar.

d'accéder aux données à lire ou à écrire. Un bus bidirectionnel de données qui véhicule les données. Et un bus bidirectionnel de contrôle qui permet la synchronisation et transmet également les signaux de réponse des éléments matériels.

L'utilisation d'une architecture avec bus impose le partage de celui-ci entre les modules. Ce partage doit être géré par un arbitre qui gère les opérations entre les maîtres (source utilisant le bus) et les esclaves (cible du maître). Beaucoup d'exemples de bus sont déjà présents comme par exemple le STBus [STM] de STMicrolelectronique et l'architecture AMBA [AMB] développée par ARM. Ce type de structure partagée présente une forte limitation de performances au niveau des échanges de données. Si le nombre de modules qui accèdes au bus est trop important, le système peut se retrouver avec un trop grand nombre d'accès au bus à gérer (goulot d'étranglement des données). De plus, ce type d'architecture limite aussi la flexibilité du réseau de communication, car une conception spécifique doit être réalisée pour chaque système.

La communication par réseau crossbar (figure 2.13.c) est réalisée par un multiplexeur de données de taille (N, M). Dans un réseau à crossbar, n'importe quel élément peut être connecté à un autre élément pour permettre plusieurs communications simultanément. Un crossbar comporte N entrées et M sorties, permettant jusqu'à (N,M) connexions. Quand deux ou plusieurs éléments essayent d'accéder au même élément, un arbitre laisse l'accès à un seul élément tandis que les autres doivent attendre. L'architecture crossbar est très coûteuse en ressources donc sa flexibilité n'est pas efficace en terme de coût en silicium.

Le dernier type de communication est réalisé par un réseau sur puce ou NoC (Networks

on Chip) [Jov09]. Ce type d'architecture permet une communication entre modules géré par un réseau. Ce type de réseau apparait comme une alternative pour connecter des modules intégrés sur une même puce vis-à-vis d'une approche d'interconnexion basée sur une structure de communication de type bus partagé ou hiérarchique. Les NoC reposent sur une paquétisation de données pour transporter les données d'une source vers une cible à travers un réseau d'aiguilleurs (routeurs) et de canaux d'interconnexion. Un réseau sur puce est conçu sur un modèle de référence OSI (Open Systems Interconnection) comportant 3 grandes couches. La première est la couche physique. Cette couche de plus bas niveau pour le NoC assure la transmission des données. La couche architecture permet la liaison des données entre les routeurs, elle est divisée en deux sous-couches. La sous-couche liaison de données dont le rôle est la détection et la correction d'erreurs et la sous-couche réseau qui gère la manière dont les paquets sont aiguillés et acheminés vers d'autres routeurs. Pour finir, la couche de haut niveau appelée application, a pour rôle la gestion des communications entre les modules du système (protocoles de communication et les adaptations d'interfaces).

Pour utiliser la reconfiguration sur une cible de type FPGA, celui-ci a besoin d'une architecture composée de différents blocs (mémoire, communication...) et d'un système de communication spécifique [OWTK10, HH08]. L'architecture est scindée en deux parties (figure 2.14). La première est dite statique (elle ne sera jamais modifiée) et la seconde est reconfigurable. La partie statique doit intégrer un manager de reconfiguration, une zone mémoire pour stocker les bitstreams de configurations et les différents IP nécessaires au système. La partie reconfigurable est divisée en plusieurs régions reconfigurables. La communication est présente à plusieurs niveaux sur ce type d'architecture. Oetken [OWTK10] explique qu'une architecture qui exploite la reconfiguration a besoin de plusieurs bus de communications. Un bus local (StaticBus) est nécessaire pour la communication entre IP statiques et un second (ReconfBus) permet de connecter les régions reconfigurables entre elles. Par conséquent, il est nécessaire d'implémenter un pont sur l'architecture pour que les données puissent transiter d'un bus à l'autre.

2.4.2 Éléments de mémorisations

Les architectures reconfigurables demandent des ressources mémoires pour réaliser les échanges entre les unités de traitement. Les performances globales des architectures sont fortement limitées par les performances des unités mémoires. Ce problème trouve une réponse dans la mise

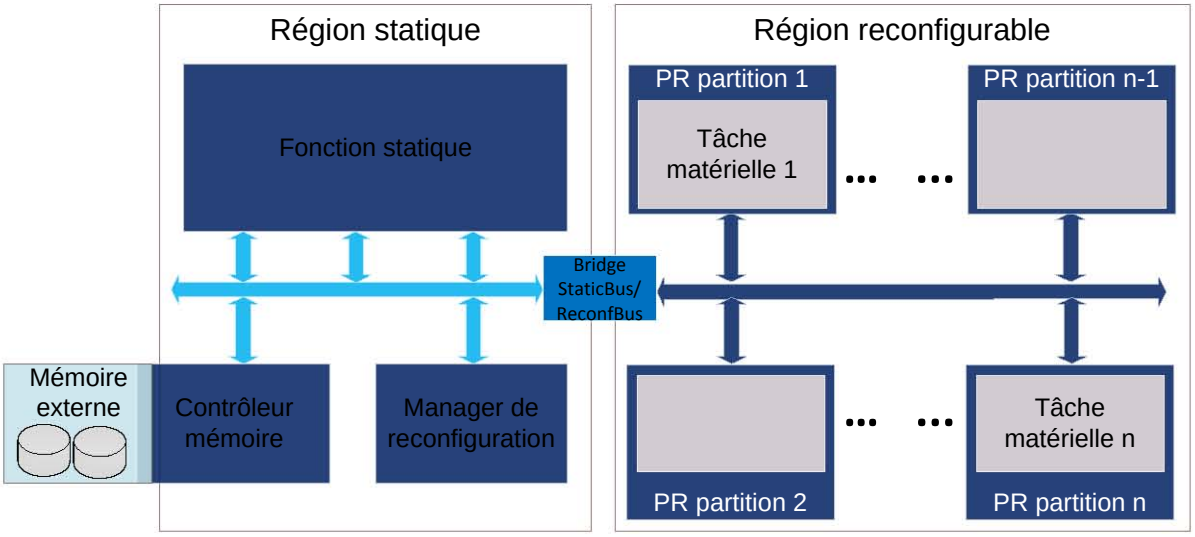


FIGURE 2.14 – Architecture d'un FPGA reconfigurable, séparation des régions statique et reconfigurable [OWTK10, HH08].

en place d'une organisation hiérarchique des mémoires qui permet de répondre aux besoins en performances des systèmes. Beaucoup de systèmes utilisent une hiérarchie à base de mémoires cache. Pour ce qui est des FPGA les constructeurs proposent actuellement des FPGA disposant de blocs mémoires de tailles de plus en plus grandes (jusqu'à 180 Mo dans les FPGA les plus récents, comme le Virtex 6 de Xilinx). Ouaiss [OV01], présente le problème de synthèse mémoire sur FPGA disposant d'une hiérarchie mémoire complexe. Il s'appuie sur le formalisme ILP (Integer Linear Programming) qui propose la prise en compte de l'ensemble des paramètres d'une mémoire comme contraintes globales (nombre de mémoires disponibles, tailles, nombres de ports d'entrées/sorties, temps d'accès). Cette formulation s'appuie sur des structures de données identifiées auparavant et donc déjà formées. Ceci implique de connaitre parfaitement le système implémenté et ses besoins en mémoire. Ce besoin montre les limites de la méthode.

2.4.3 L'aspect interface

Murgida [MPR+06] détaille dans son étude les interfaces des zones reconfigurables. Il définit une zone reconfigurable non configurée comme une « black box » vide. Cette black box est définie pendant la phase de conception de l'architecture. Le concepteur devra fixer les entrées et les sorties (BUS ou Port) de cette zone. Par conséquent les interfaces de communication d'une région reconfigurable sont statiques, seule la fonction IP core (PR module) est reconfigu-

rable. De plus l'IP devra être compatible avec l'interface de la zone reconfigurable. Pour cela, il explique comment générer un IP core compatible avec une zone reconfigurable.

Sa méthode contient trois phases, la première phase est la description VHDL de l'IP. Les deux phases qui suivent servent à mettre en place l'interface entre l'IP et la zone reconfigurable. Pour cela, il propose de créer une fonction « Writer » pour écrire et une fonction « Reader » pour lire, interpréter et sauvegarder toutes les informations nécessaires aux prochaines étapes. Les différents modules (le core, l'interface, le Reader et Writer) sont ensuite rassemblés pour créer l'IP core qui sera implémenté sur la « black box ». Murgida a montré qu'il était nécessaire de créer des interfaces pour les zones reconfigurables, il propose une première exploration pour optimiser ces interfaces, mais il ne donne pas de solution pour rendre cette interface générique compatible avec les différents IP qui existent.

Huang [HH08] reprend le concept d'interface en utilisant le terme wrapper, il propose un wrapper pour encapsuler une IP hardware. Pour que son wrapper soit compatible avec les différents IP hardwares il est nécessaire de le rendre commun à toutes les IPs. Il faut donc modifier les différents IP pour les rendre compatibles avec le wrapper. Ce wrapper permet de recevoir et d'envoyer des données comme celui de Murgida [MPR+06]. La méthode de Huang est une bonne approche théorique du système, mais elle nécessite un effort considérable pour adapter les IP aux wrappers. Le concepteur doit dans un premier temps créer une interface statique qui sera commune à toutes les zones reconfigurables. Cette interface sera capable de communiquer avec le wrapper. Ensemble, ils permettront la communication avec les autres IP du système par l'intermédiaire d'un bus. Ces deux modules permettront aussi de gérer les reconfigurations grâce à un contrôleur de swap.

Ce contrôleur permet de connaître l'état de l'IP et de savoir si elle peut ou ne peut pas être reconfigurée. Le buffer de contexte lui permet de sauvegarder les informations sur les données entre deux reconfigurations. Une fois l'interface et le wrapper réalisés il faut adapter l'IP pour la rendre compatible avec les modules de communications. Il faut rajouter une interface de donnée ainsi qu'une machine d'état pour connaître les états de l'IP. Ces deux modules doivent être réalisés pour chaque IP. Voyant l'effort à fournir, Huang propose de simplifier le wrapper. Il utilise un « LISS wrapper » spécialement conçu pour fonctionner avec un bus OPB IPIF (Xilinx) [XM]. L'interface du BUS remplace l'interface statique du wrapper. Il supprime la sauvegarde du contexte du wrapper, et déclare l'IP comme un esclave. Il propose de remettre à zéro (reset) l'IP entre deux reconfigurations. Par conséquent, il n'est plus nécessaire de sauvegarder le contexte.

Cette solution est intéressante, mais elle ne permet pas de traiter des flux de données constants et oblige le concepteur a utiliser un BUS OPB.

VAPRES [JBGR10] (figure 2.15) est une architecture SoC FPGA multifonction partiellement reconfigurable composée d'un processeur soft de type Microblaze implémenté dans la région statique relié à un ensemble de régions reconfigurables (PRR). L'architecture VAPRES est composée de deux régions fondamentales : la région de contrôle et la région de traitement de données. VAPRES introduit plusieurs nouvelles fonctionnalités architecturales pour répondre aux contraintes d'une conception reconfigurable, telles que la capacité de faire fonctionner des modules matériels à des fréquences d'horloge indépendante et configurable (domaines d'horloge locale). Un ensemble de fonctions API fournissant la fonctionnalité du système de bas niveau est proposé. Cette architecture utilise aussi un module d'interface pour chacune de ces zones reconfigurables. Du point de vue du haut niveau, chaque interface à plusieurs ports de sorties et d'entrées. En interne, une interface de commutation est constituée d'un ensemble de multiplexeurs et d'un registre. Pour établir un canal de diffusion, le microprocesseur est utilisé. Après l'établissement du canal, des mots de données sont échangés entre le processeur et l'interface au moyen des registres. Les interfaces de communication sont équipées de deux FIFO (une d'entrée et une de sortie).

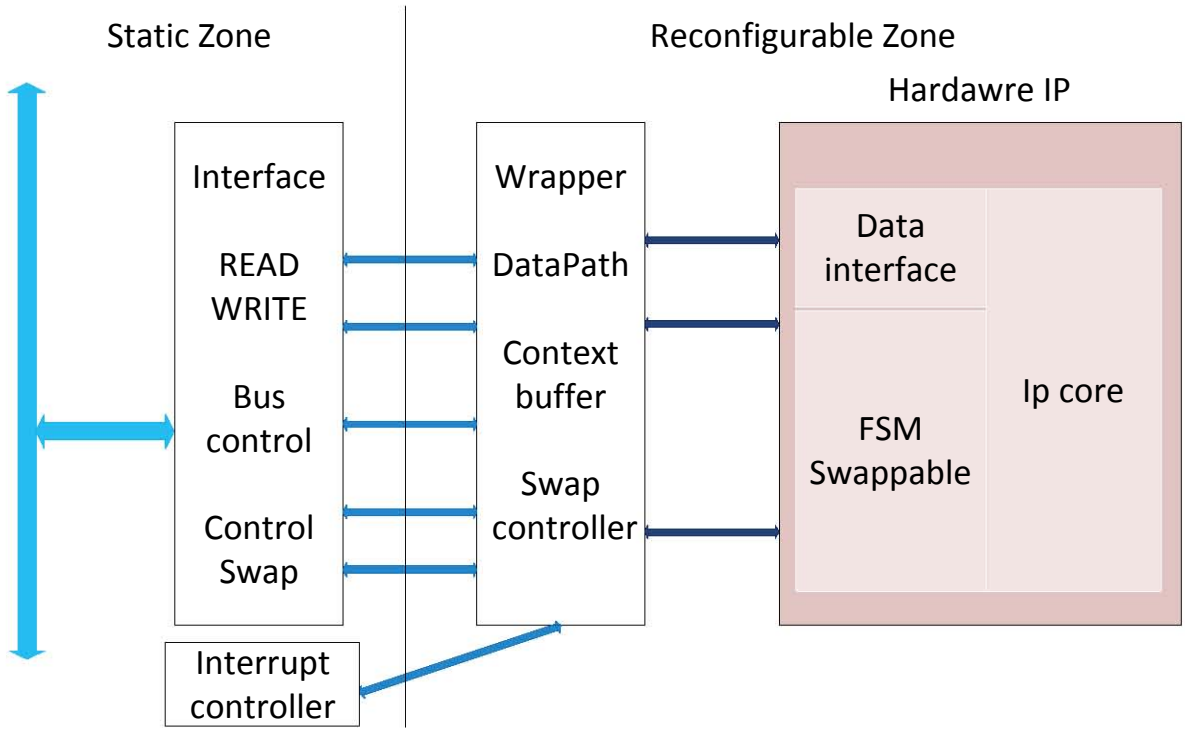


FIGURE 2.15 – Architecture avec partitionnement statique et reconfigurable [JBGR10].

2.5 Conclusion

Ce chapitre nous a permis de présenter les techniques et les méthodes qui permettent de reconfigurer les FPGA. Nous avons pu voir aussi que les interfaces d'interconnexions des régions reconfigurables ont évolué avec les flots de conceptions. Par contre, les FPGA ne permettent pas encore l'adaptation de la taille des régions reconfigurables.

Il est donc nécessaire d'explorer de nouvelles méthodes d'adaptation pour placer des tâches matérielles de taille différentes sur des régions reconfigurables en respectant les contraintes sur les éléments de communication. Avant de présenter notre méthode, nous allons dans le chapitre suivant expliquer les principes de l'adaptation des partitions reconfigurable aux modules reconfigurables, un état de l'art sur les travaux existant sera aussi présenté.

Chapitre 3

Adaptation des partitions reconfigurables aux modules reconfigurables

Sommaire

3.1 Introduction

Les systèmes adaptatifs et reconfigurables permettent d'implémenter plusieurs fonctions sur une même région. Grâce à la reconfiguration dynamique partielle, on peut supprimer certaines fonctions (tâches matérielles ou IP) du dispositif FPGA et charger de nouvelles fonctions à leur place. La question de la communication est l'un des défis dans la mise en place des tâches matérielles sur les régions reconfigurables. Chaque tâches matérielles qui devient un module reconfigurable doit avoir une interface de communication commune avec la région reconfigurable. L'autre défi qui est sûrement le plus complexe et le plus intéressant à résoudre est la possibilité d'utiliser au maximum les ressources d'une région reconfigurable et la possibilité d'augmenter la flexibilité du FPGA. Habituellement, les architectures reconfigurables partielles sont conçues avec des régions PR séparées et dimensionnées selon la plus grande tâche matérielle (PR module) implémentée dans celle-ci. L'inconvénient de cette approche est la faible utilisation des ressources, car même un petit module PR occupe une région PR complètement. L'autre inconvénient est qu'il n'est pas possible d'utiliser deux petites régions PR pour implémenter un module de plus de plus grande.

Pour résoudre cette problématique, deux solutions peuvent être envisagées, soit la possibilité d'adapter les tâches matérielles en partitionnant les IPs, ou la possibilité d'adapter les régions PR aux tâches matérielles. La première solution est réalisée en modifiant le code source de l'IP pour la partitionner en sous IP, cette méthode n'est pas toujours possible, car nous ne disposons pas forcément des codes sources des IP. Dans ce chapitre, nous allons présenter ces deux méthodes et déterminer leurs avantages et leurs inconvénients.

3.2 Principe d'adaptation

Pour expliquer les méthodes d'adaptation, nous utiliserons l'exemple de la figure 3.1. Considérons un système qui nécessite l'utilisation de deux modules ALPHA et BETA de tailles différentes. ALPHA utilise environ trois fois plus de ressource que BETA. Généralement, pour pouvoir placer les deux tâches dans la même région PR, il faut que la taille de cette dernière soit supérieure ou égale à la plus grande tâche matérielle (ici ALPHA). Par conséquent, quand le module PR ALPHA sera placé sur la région PR, il utilisera 100 % des ressources disponibles.

Par contre, quand le module PR BETA sera placé, il n'utilisera qu'un tiers des ressources, les autres ressources seront perdues et l'efficacité en surface seras dégradée.

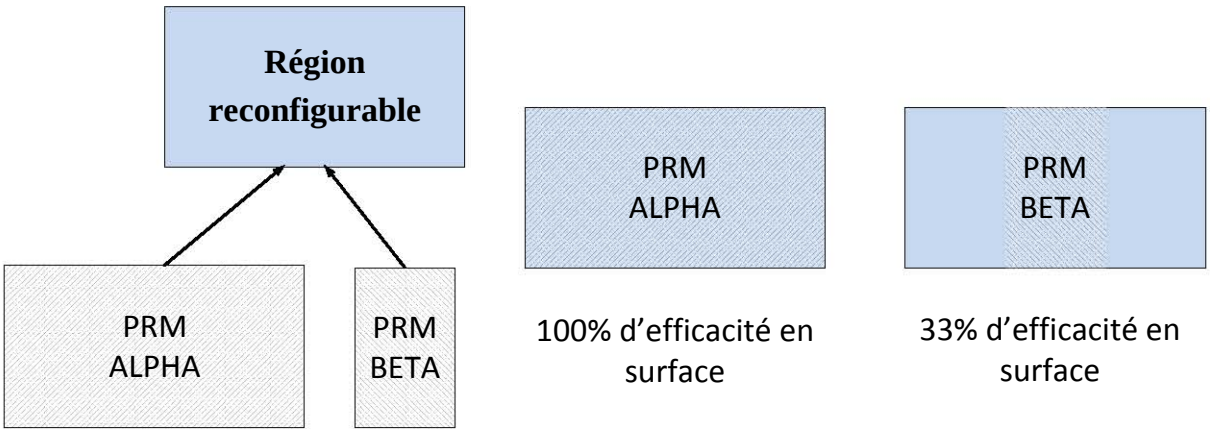


FIGURE 3.1 – Exemple d'application qui nécessite l'adaptation de la région reconfigurable avec un partitionnement en partitions PR.

3.2.1 Adaptation des tâches matérielles

L'adaptation de la taille d'une tâche matérielle à une région reconfigurable consiste à la partitionner en sous-tâches. C'est une technique similaire au partitionnement d'un graphe flot de données et du placement temporel. Chaque nouvelle sous tâche est placée sur une région reconfigurable. Dans l'exemple présenté ci-dessus, pour augmenter l'efficacité en surface, il faut partitionner la tâche ALPHA en deux sous-tâches, une de taille de la tâche BETA et l'autre de taille égale à la différence entre ALPHA et BETA. La figure 3.2 nous montre que grâce à cette méthode de partitionnement des tâches, l'efficacité en surface des deux régions PR est maximale. De plus, quand la tâche BETA est placée, la région PR non utilisée devient libre et on peut y placer d'autres tâches.

En théorie, cette méthode permet d'obtenir de très bons résultats avec une efficacité en surface proche de 100 %. Par contre, en pratique le partitionnement des tâches matérielles est très complexe. Cette problématique de partitionnement des tâches a largement été étudiée pour le placement temporel [CHW00, Car03]. Les méthodes existantes ont montré leur efficacité dans le cas de graphes flot de données simples, mais resteny inefficaces et pratiquement impossible à mettre en œuvre pour la plupart des tâches matérielles complexes.

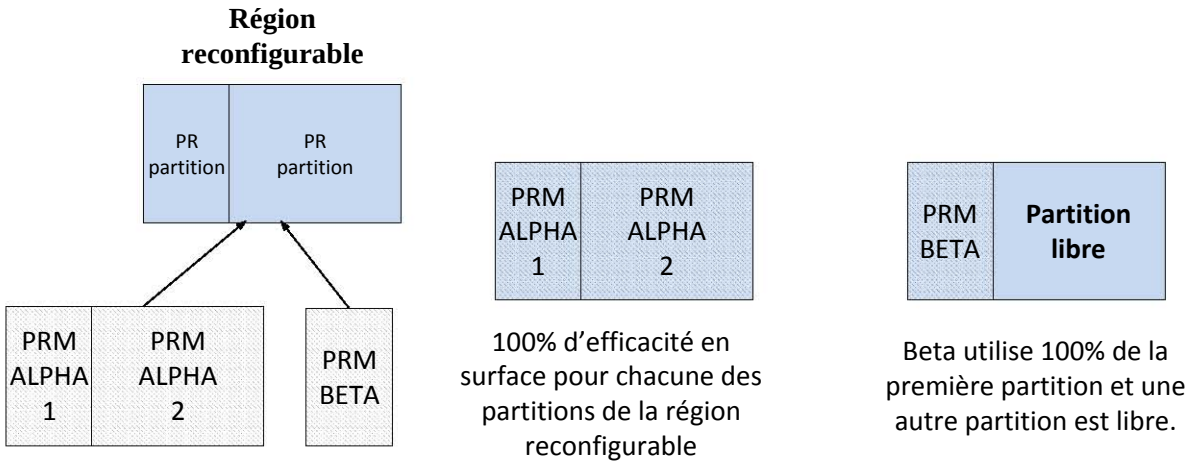


FIGURE 3.2 – Exemple de partitionnement d'une région reconfigurable.

3.2.2 Adaptation des régions reconfigurables

Une autre méthode consiste à diviser les régions reconfigurables en plusieurs partitions partiellement reconfigurables (partition PR). Avec cette méthode, les tâches matérielles ne sont plus modifiées, elles gardent leur taille d'origine. Par contre la région reconfigurable est partitionnée en N partitions qui permettent d'accueillir n'importe quel module PR. Chaque module PR recouvre une ou plusieurs partitions reconfigurables en fonction de sa taille. Dans l'exemple présenté ci-dessus, il fallait partitionner l'IP ALPHA pour obtenir une nouvelle IP de la taille de BETA. Avec cette méthode, les IP ne sont plus partitionnés, mais c'est la région reconfigurable qui est partitionnée. La figure 3.3 nous montre que grâce à cette méthode de partitionnement de la région reconfigurable, on obtient une efficacité proche de 100 % pour chaque partition reconfigurable dans les deux cas d'implémentations. De plus, quand le module PR BETA est implémenté, nous récupérons plusieurs partitions PR qui peuvent être utilisées pour implémenter d'autres modules PR.

Ce principe de partitionnement de la région reconfigurable est avantageux par rapport à la méthode précédente, car il ne demande pas au concepteur de modifier les IP. De plus, cette méthode permet une grande flexibilité d'utilisation pour la région reconfigurable. Dans le paragraphe suivant, nous allons présenter les travaux existants dans lesquelles cette problématique a été traitée.

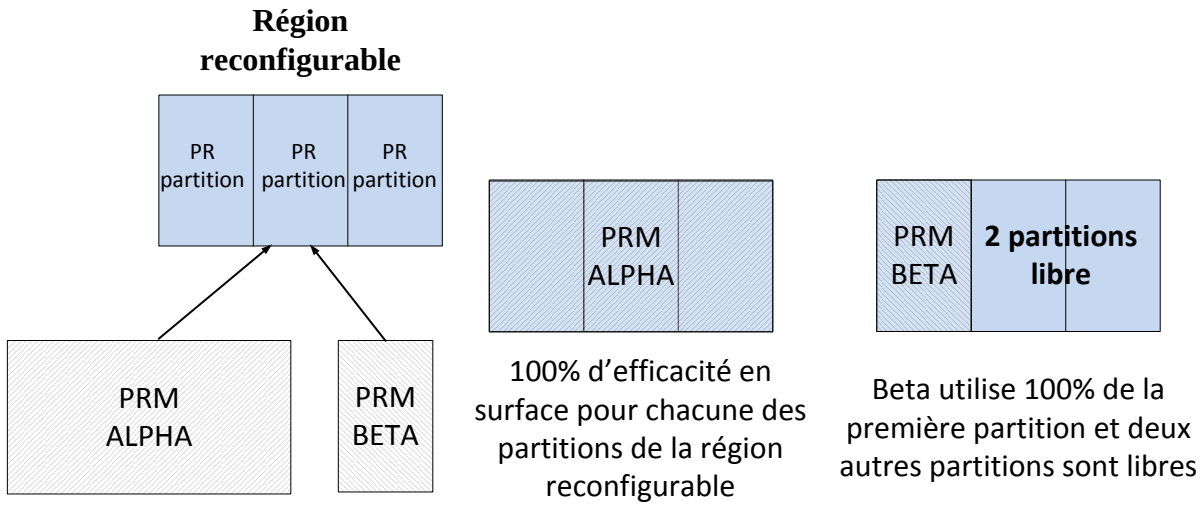


FIGURE 3.3 – Exemple de partitionnement d'une région reconfigurable.

3.3 Travaux existants

3.3.1 Structure monodimensionnelle basée sur un Bus

En 2004 Herbert Walder et Marco Platzner ont proposé un découpage de la région reconfigurable en slots de tailles fixes [WP04]. Chaque slot représente une partition PR utilisée pour la mise en place d'un module PR avec une taille inférieure ou identique à la partition (figure 3.4). Une infrastructure de communication spécifique basée sur les bus macros est ajoutée pour établir un lien entre les différents modules PR et le reste du système.

Le système de communication est un bus constitué de fils de connexion, de bus macros et de deux arbitres de bus (BARL, BARR) situés à droite et à gauche de l'architecture dans les zones réservées au système d'exploitation. De plus, un contrôleur d'accès au bus (BAC) est placé dans chacune des tâches utilisateurs (module PR). Le bus est divisé en trois parties, le bus de gauche, le bus de droite, et le bus de commande. Le bus de gauche sert de communication pour la lecture et l'écriture des données situées dans les tâches de gauche et le bus de droite accède aux éléments des tâches de droite. Le bus de gauche est arbitré par le BARL et le bus de droite par la BARR. Le bus de commande permet de faire des requêtes aux différents arbitres. Le contrôleur d'accès au bus (BAC) fournit une interface qui permet la communication entre la tâche utilisateur et le reste du système.

Par la suite, Herbert Walder, Samuel Nobs, et Marco Platzner ont proposé une implé-

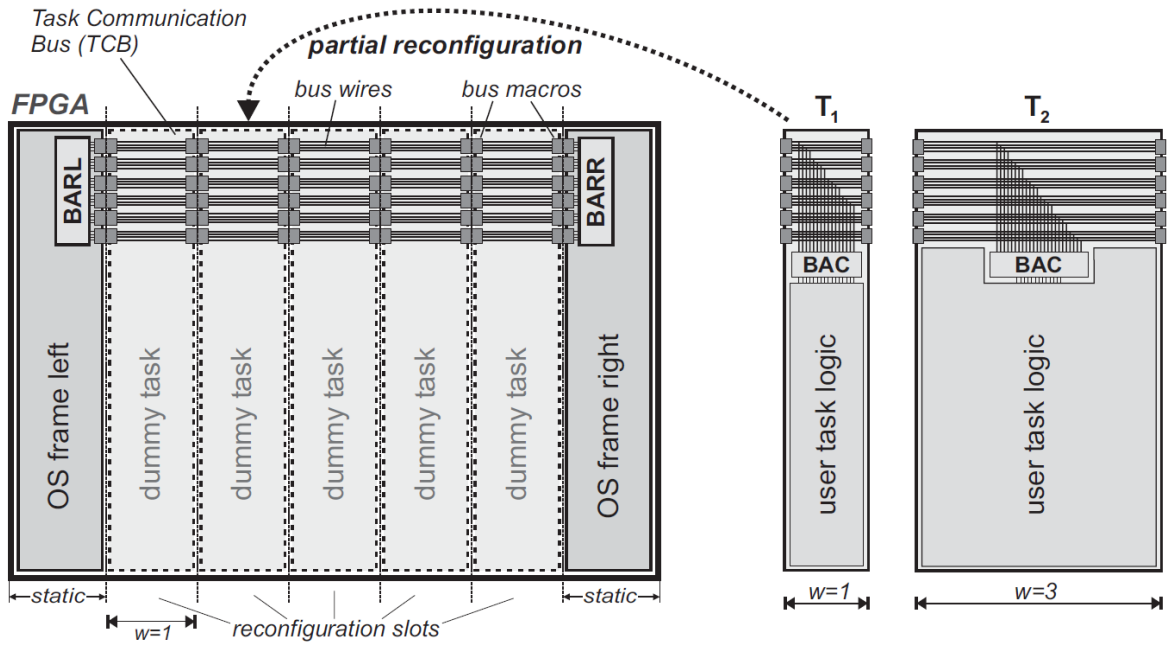


FIGURE 3.4 – Architecture reconfigurable utilisant une communication par bus [WP04].

mentation [WNP04]. Ils ont utilisé une plate-forme de prototypage pour les systèmes reconfigurables, la XF-BOARD qui dispose d'un VIRTEX 800. Ce type de FPGA offrait des possibilités pour les architectures reconfigurables mais elles étaient limitées à une reconfiguration d'une colonne CLB complète. Cette implémentation avec un placement sur une dimension a permis de valider leur système de communication entre les partitions PR mais ils n'ont pas mis en œuvre la possibilité d'implémenter des modules PR de tailles différentes.

3.3.2 Structure bidimensionnelle basée sur un bus

En 2005, Michael Hübner, Christian Schuck et Jürgen Becker [HSB06b] ont présenté un placement sur deux dimensions. Leur architecture pour la reconfiguration partielle en 2-D se base sur le flot de conception de Xilinx. Ils ont choisi d'ajouter des macros de routage pour mettre en œuvre les éléments de communications. Ces macros permettent de mettre en œuvre la communication des régions PR grâce à l'ajout de bloc de routage. L'ajout de ces éléments est réalisé par la manipulation du bitstream. L'approche introduite par les auteurs permet de démontrer que le placement de modules variables est possible sur une architecture reconfigurable qui utilise les FPGA Virtex 2. Cette approche qui utilise la modification de bitstream a aussi été

validée par Miguel L. Silva et Joao Canas Ferreira [SF08] mais l'obligation de modifier tous les bitstreams de reconfiguration est très fastidieuse.

Par la suite en 2007, Jens Hagemeyer et ses collègues de l'université de Paderborn en Allemagne ont envisagé la possibilité d'intégrer une nouvelle interface de communication dans les régions reconfigurables [HKKP07a, HKKP07b]. Ils ont présenté un nouveau flot de conception pour la conception et la mise en œuvre d'architectures reconfigurables partiellement. Ce flot commence à partir de la spécification des composants du système et le partitionnement du système et se termine par la génération des bitstreams de reconfiguration. Ils proposent un modèle à plusieurs couches pour mettre en œuvre des protocoles de communication normalisés (par exemple, AMBA ou Wishbone) dans un système partiellement reconfigurable. Le modèle divise l'infrastructure de communication dans différents niveaux d'abstraction. Ainsi, des IP avec une interface de communication normalisée peuvent être intégrées dans un système partiellement reconfigurable. Un exemple d'implémentation de l'infrastructure de communication utilisant le protocole Wishbone est présenté. Comme dans les articles précédents ils n'ont pas mis en œuvre la possibilité d'implémenter des modules PR de taille différentes.

3.3.3 Structure bidimensionnelle basée sur Noc

3.3.3.1 Architecture

En 2009 Thilo Piontek et ses collègues de l'université de Lübeck en Allemagne [PKAM09] proposent une solution pour l'adaptation des régions PR. Pour réaliser cette solution, l'auteur propose une nouvelle architecture reconfigurable [PAKM09]. La structure de base de l'architecture reconfigurable est présentée dans la figure 3.5. Cette architecture prend en compte les possibilités de reconfiguration et les restrictions physiques des FPGA de Xilinx de la série Virtex-4. La zone logique du FPGA est divisée en une région statique et une région reconfigurable qui est à son tour subdivisée en une grille de partitions de tailles égales. La logique de commande de la reconfiguration y compris le port de configuration interne (ICAP), les contrôleurs d'entrées et de sortie, le contrôleur du système et toutes les fonctions statiques résident dans la partition statique.

Le réseau de communication de la région reconfigurable est basé sur une topologie adaptative de types NoC [PKA06]. Le routage est effectué au moyen de tables de routage mémorisées

localement fournies par une unité de commande globale. Quatre différents types de blocs de construction de base peuvent être mappés sur les partitions reconfigurables : des liens de communication horizontale et verticale (Type H et V), des commutateurs de réseau (Type S), et des blocs spécifiques à l'application (Type 0). Chaque bloc fournit, à la même position, des liaisons de communication à des blocs adjacents de sorte que chaque bloc peut être connecté avec l'autre. Les liaisons qui permettent la communication sont constituées de bus macros qui sont des lignes de communication qui franchissent la frontière entre deux blocs. Lors de l'exécution, les blocs sont mappés sur des partitions et ils peuvent être ensuite échangés. Cela permet de mettre en place une structure adaptative.

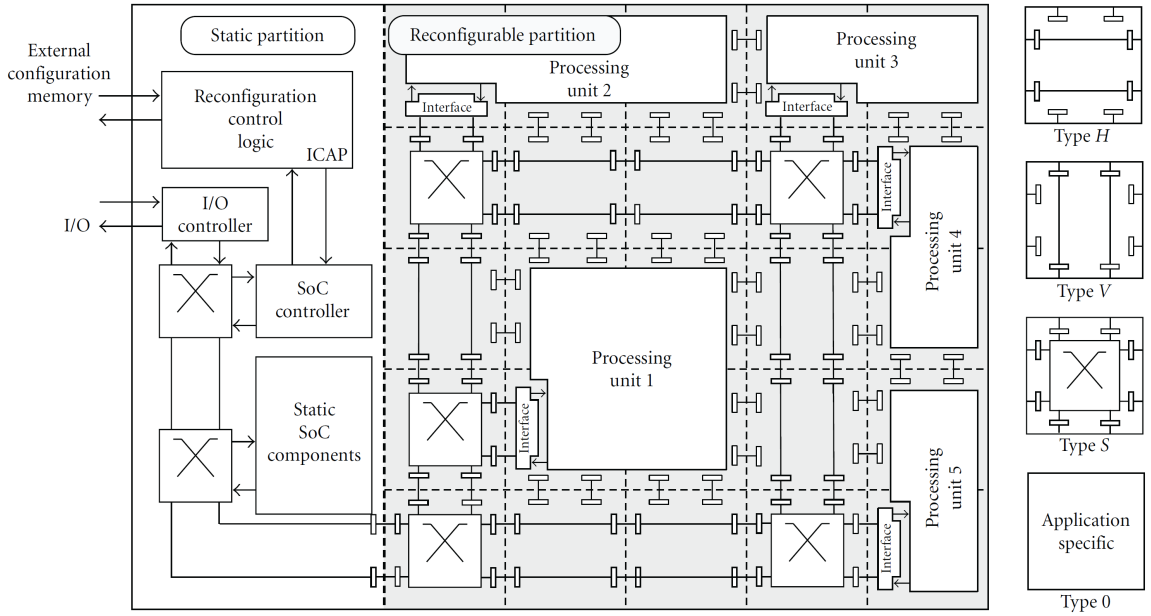


FIGURE 3.5 – Structure de base d'une architecture reconfigurable pour l'adaptation des régions PR [PKAM09].

3.3.3.2 Flot de conception

L'architecture présentée est une base de travail pour l'adaptation des partitions PR aux modules PR mais dans le flot de conception EAPR (early access partial reconfiguration), la taille et l'emplacement des partitions PR et des modules PR doivent être définie une fois pour toute. Par conséquent, la totalité du flot doit être exécutée à nouveau lorsque le nombre, la taille ou l'emplacement des partitions PR sont modifiées. Cette nouvelle exécution du flot de conception

ne garantit pas la correspondance des interfaces de communication avec la première exécution. Pour détourner les contraintes liées au flot de conception EAPR, Thilo Piontek et ses collègues ont proposé, un nouveau flot de conception illustré par la figure (figure 3.6). La technique qu'ils proposent permet d'avoir une correspondance exacte au niveau des interfaces de communication entre la partie statique et la région reconfigurable. En supposant que deux ou plusieurs conceptions partagent les mêmes éléments statiques et les mêmes interfaces entre la partie statique et les régions reconfigurables, l'idée de base qu'ils proposent est de mettre en œuvre les éléments statiques communs dans une première conception puis de générer la Hard Macro (Annexe 1) correspondante qui sera utilisée dans une nouvelle conception. La figure 3.6 présente la méthode de conception proposé. Le flot comprend cinq étapes et utilise les netlists des différentes conceptions.

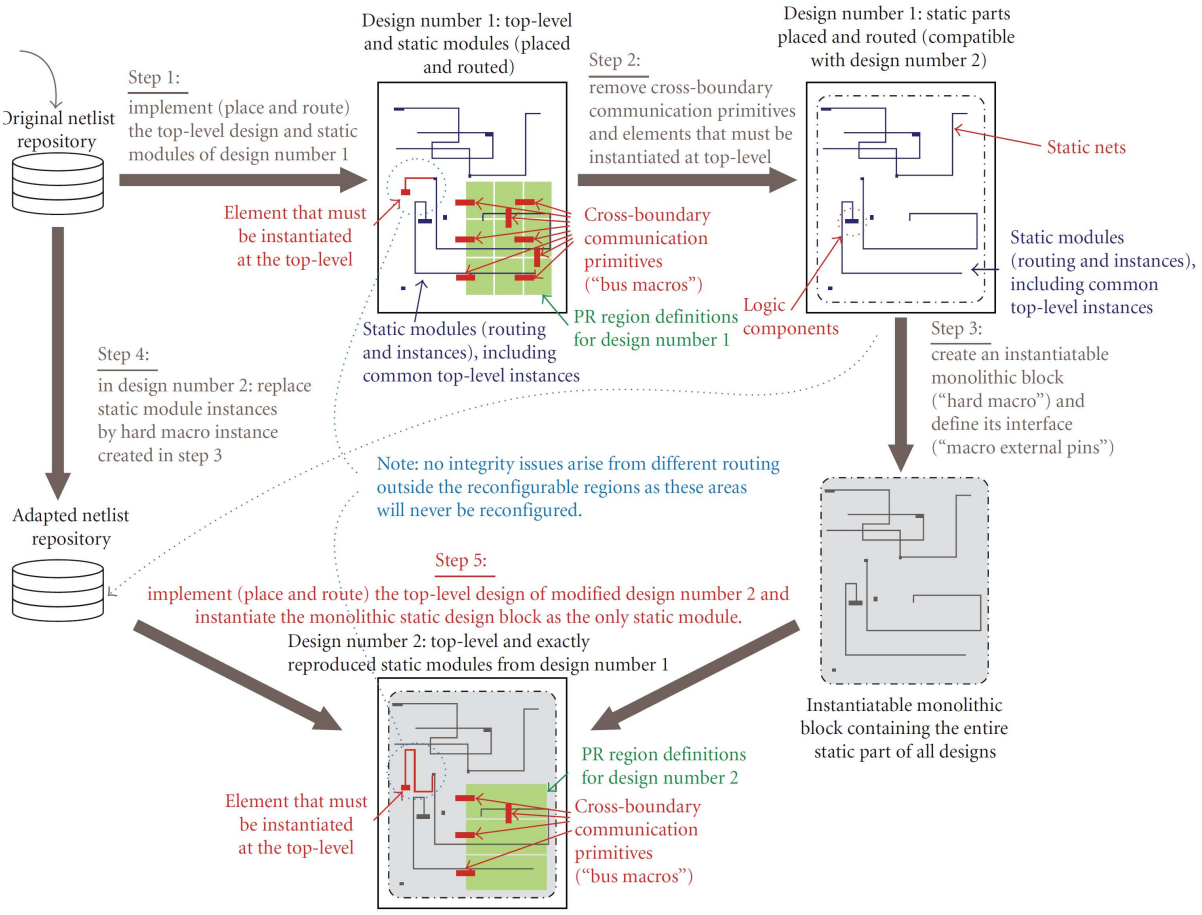


FIGURE 3.6 – Flot de conception en cinq étapes pour l'adaptation des régions reconfigurables avec des Hard Macro [PKA06].

La technique de conception présentée ouvre la voie pour des conceptions partiellement reconfigurables qui utilisent des modules PR de différentes tailles avec des outils de conception standard. L'inconvénient de la technique proposée est que la conception et la mise en œuvre doivent être éditées manuellement dans un éditeur de netlist pour ensuite pouvoir créer une Hard Macro. De plus, cette méthode force le concepteur à réimplémenter la totalité du projet pour chaque nouvelle conception avec une région reconfigurable partitionnée différemment.

3.3.4 Structure à base d'un bus reconfigurable

3.3.4.1 Architecture

La même année Andreas Oetken, avec ses collègues de l'Université de Erlangen-Nuremberg en Allemagne et Dirk Kocho de l'Université d'Oslo en Norvège présentent une architecture qui permet un placement flexible des modules reconfigurables sur un FPGA [OWTK10]. Il présente une conception qui permet de fournir une nouvelle infrastructure de communication pour un système reconfigurable. Pour réaliser cette infrastructure de communication, il propose une architecture reconfigurable qui utilise des bus de communications spécifiques.

L'architecture (figure 3.7) comprend un CPU de type Power PC, des IP statiques et des régions reconfigurables. Les régions reconfigurables sont traversées par un bus de communication de type ReCoBus [KHT08]. ReCoBus est un bus qui permet à des modules d'être connectés à différentes interfaces avec un nombre variable d'emplacements. Ce bus est implémenté sous la forme d'une Hard Macro pour son implémentation. La taille du bus peut être augmentée en fonction du nombre de slots que le module reconfigurable utilise. En plus du bus ReCoBus des connexions de type entrées/sorties (I/O) sont disponibles. Elles permettent de lire les données et de les transférer aux modules reconfigurables suivants (les deux partitions reconfigurables par l'intermédiaire des I/O doivent être l'une à côté de l'autre. Les modules reconfigurables doivent obligatoirement être réalisés avec l'outil ReCoBus-Builder [KBT09].

3.3.4.2 Flot de conception

Cette architecture a nécessité l'utilisation d'un flot sur mesure. Le flot est divisé en quatre à cinq étapes, selon le type de construction (statique ou partielle). Le flot génère des hard macros pour les ReCoBus. Cette architecture et son flot de conception permettent d'avoir une

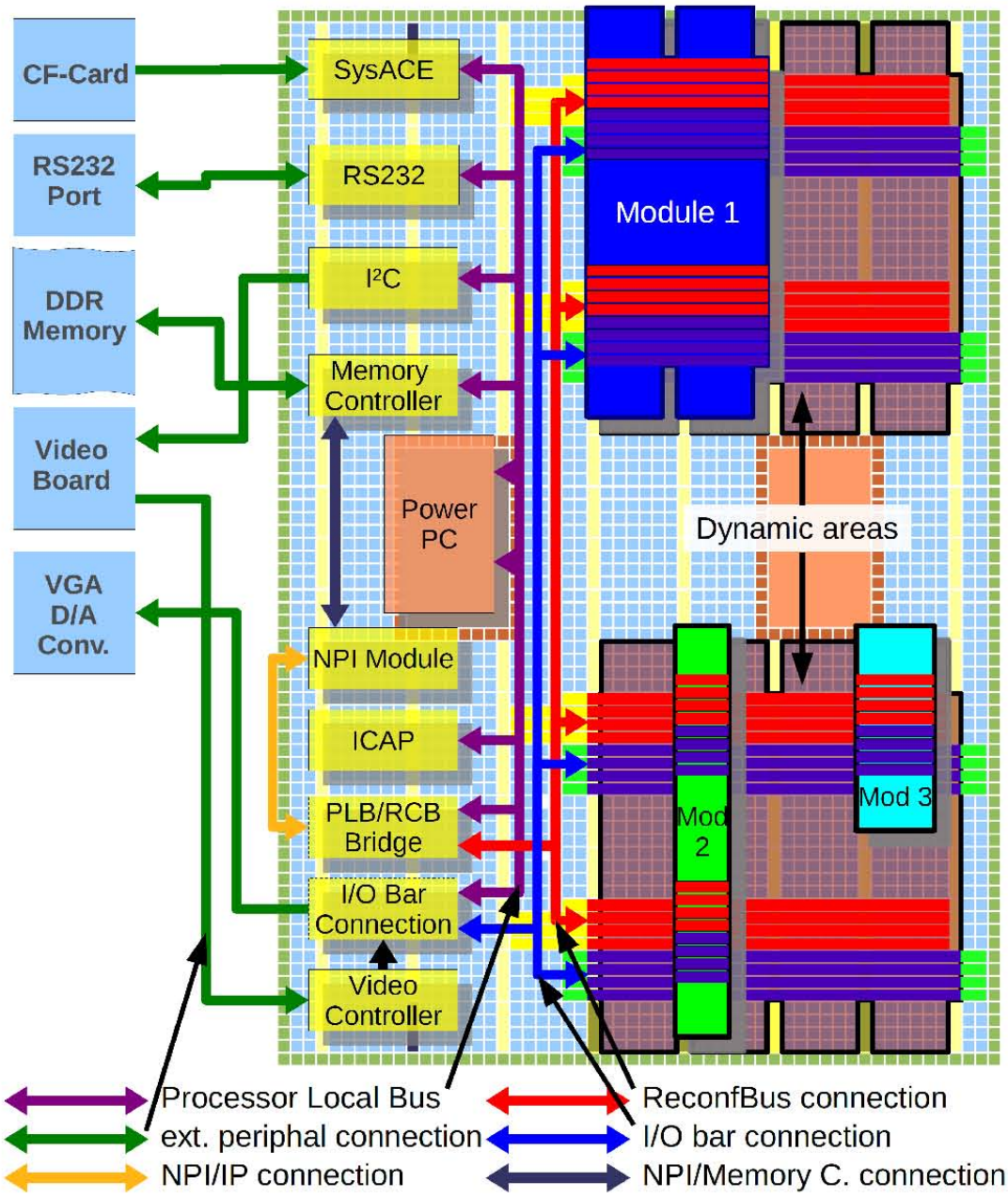


FIGURE 3.7 – Architecture reconfigurable utilisant une communication de type ReCoBUS [OWTK10].

architecture reconfigurable flexible, par contre elle oblige le concepteur à utiliser un bus de communication de type ReCoBus et à créer une Hard Marco pour réaliser l'implémentation de l'architecture.

3.4 Discussion

La reconfiguration dynamique des FPGA est donc une solution pour les systèmes auto adaptatifs, mais cette solution doit être améliorée pour augmenter ses possibilités en matière d'adaptation. Il est donc nécessaire d'explorer de nouvelles techniques pour permettre le placement de tâches matérielles de tailles différentes sans diminuer l'efficacité en surface du système. La solution retenue se bases sur le partitionnement de la région reconfigurable. La région reconfigurable est divisée en plusieurs partitions PR. La taille initiale de ces partitions PR est définie en fonction des besoins du système et chaque partition doit être capable d'accueillir les modules PR les plus petits. Pour le placement des modules PR les plus grands, un ensemble de PR partitions est associé. La figure 3.8 montre un exemple d'une région reconfigurable divisée en plusieurs partitions PR. Chaque module PR recouvre une ou plusieurs partitions PR en fonction de sa taille.

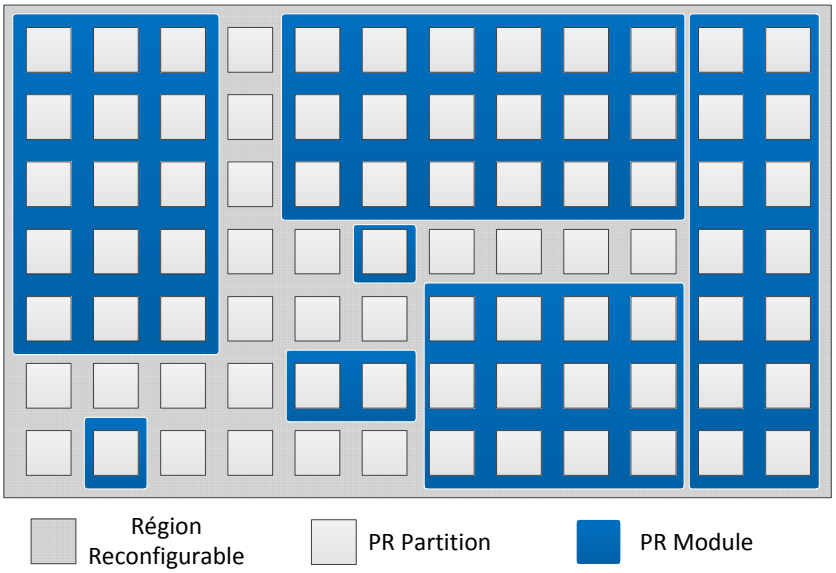


FIGURE 3.8 – Principe de l'adaptation des modules PR aux partitions PR.

Nous avons vu que le placement des modules PR de tailles supérieures à une partition PR nécessite l'association de plusieurs partitions. Cette association n'est pas autorisée dans les différents flots de conceptions (MBPR/DBPR, EAPR et PBPR). Ces différents flots de conception obligent l'utilisateur à définir une architecture de base et à définir la taille des régions reconfigurables au début de la conception. De plus, les régions reconfigurables ne peuvent pas être

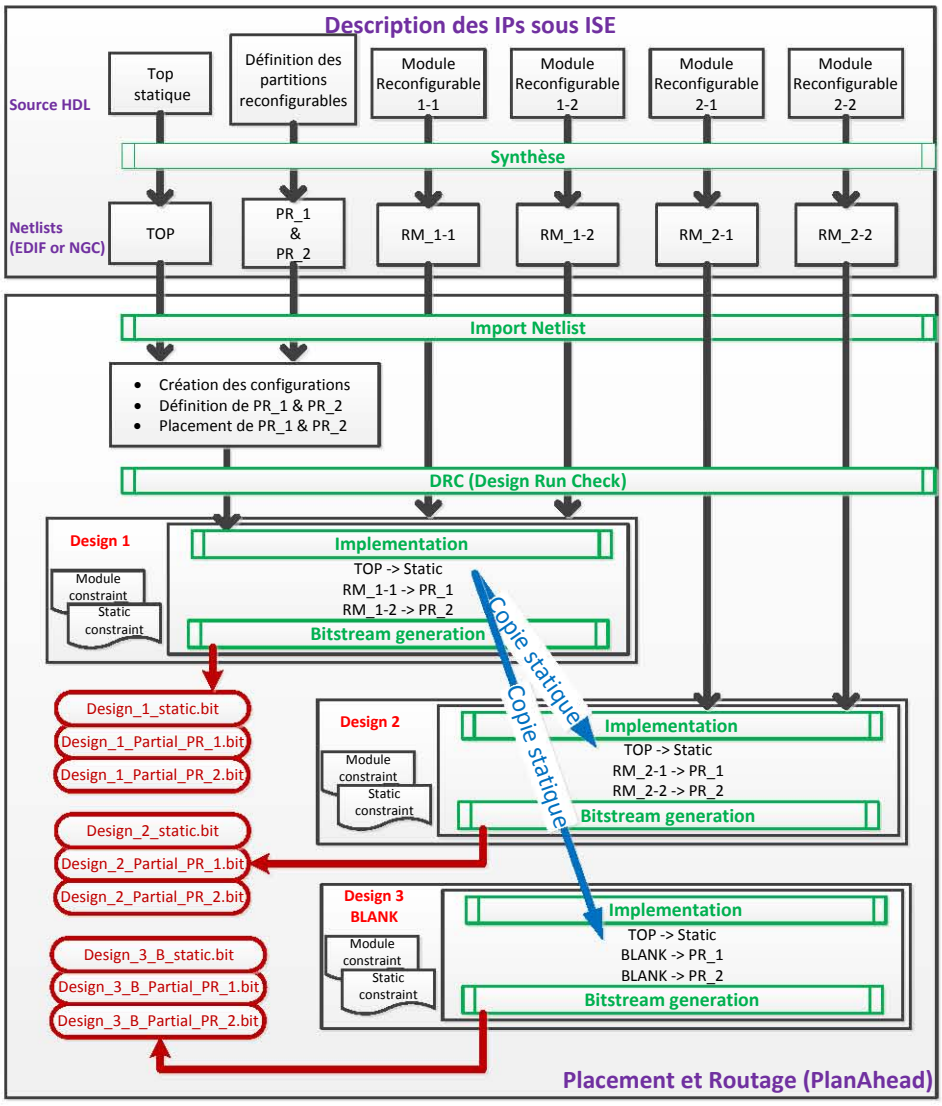


FIGURE 3.9 – Flot de conception Xilinx pour les architectures reconfigurables.

divisées en partitions. Le flot de conception PBPR est le plus récent (figure 3.9), il permet la définition de plusieurs régions reconfigurables qui sont définies au début du projet sous forme de netlist. Ces netlist ne peuvent plus être modifiées par la suite pour associer les différentes régions ou pour créer des partitions PR supplémentaires. Il est donc nécessaire d'étudier de nouvelles solutions pour surmonter ces restrictions qui ne permettent pas la création des partitions PR et la possibilité de recouvrir plusieurs partitions par un module PR. Cette solution devras être capable d'assurer la communication entre les différentes régions.

3.5 Conclusion

Dans ce chapitre, nous avons vu les principes de l'adaptation des régions reconfigurables. Deux méthodes ont été présentées, l'adaptation des tâches matérielles et l'adaptation des régions reconfigurables. Le second principe utilise des techniques de partitionnement de la région reconfigurable qui est avantageux par rapport au premier principe, car il ne demande pas au concepteur de modifier les IP. De plus, cette méthode permet une grande flexibilité d'utilisation pour la région reconfigurable.

Comme nous l'avons vu tout au long de ce chapitre le placement, la taille et l'interconnections des régions reconfigurables sont des points très importants dans la création d'une architecture reconfigurable avec une région partitionnée. Actuellement, avec les flots de conception connu il n'est pas possible d'adapter totalement ces trois critères, on peut donc conclure que la région reconfigurable n'est pas totalement flexible. Par conséquent, il est nécessaire de proposer de nouvelles méthodes qui permettront d'augmenter la flexibilité des FPGA et principalement une méthodologie et une architecture autoadaptative pour le placement de tâches matérielles de tailles variables sur des partitions reconfigurables. Cette méthodologie et cette architecture devront prendre en compte les problèmes d'interconnexion des régions reconfigurables et la technologie des pins partitions. Le but n'étant pas d'augmenter la charge des concepteurs, le nouveau flot de conception doit être compatible avec les outils existants.

Dans le chapitre suivant, une nouvelle approche méthodologique basée sur les flots de conception existants est proposée.

Chapitre 4

Méthode proposée

Sommaire

4.1 Introduction

Dans la mise œuvre de la reconfiguration dynamique, l'optimisation des ressources qui permet d'augmenter l'efficacité est un des gains principaux attendus par le concepteur. Ce gain est obtenu grâce à l'utilisation d'une région reconfigurable sur laquelle plusieurs tâches matérielles pourront être implémentées. Dans la majorité des systèmes reconfigurables, la taille d'une région reconfigurable est fixe. Dans le cas où les différentes tâches matérielles implémentées ont des tailles différentes, la région reconfigurable devra avoir une taille suffisamment grande pour être capable d'accueillir la plus grande des tâches. Par conséquent, la tâche matérielle la plus grande utilise la totalité des ressources disponibles de la région reconfigurable. Pour les tâches matérielles de tailles inférieures implémentées sur la même région reconfigurable, l'utilisation des ressources n'est plus optimisée vu que le pourcentage d'utilisation des ressources de la région reconfigurable dépendra de la taille du module reconfigurable implémenté sur celle-ci. Ce type d'implémentation implique une perte de ressource qui ne peut être utilisée par une autre tâche matérielle. De cette contrainte découle une problématique générale qui est : Comment optimiser le placement des tâches matérielles de tailles différentes sur les régions reconfigurables ? Nous avons vu dans le chapitre précédent que la meilleure solution était de partitionner les régions reconfigurables en partitions reconfigurables. Ensuite, chaque tâche reconfigurable utilise une ou plusieurs partitions reconfigurables en fonction des ressources dont elle a besoin.

Dans ce chapitre, nous présentons et formulons une nouvelle méthodologie de l'adaptation du placement des tâches matérielles de tailles différentes sur des partitions reconfigurables pour répondre à la problématique citée ci-dessus. Notre objectif est d'apporter une contribution méthodologique sur les techniques d'implémentation, qui complète les flots de conception déjà existants.

La suite de ce chapitre est organisée de la manière suivante : Dans un premier temps, les besoins pour obtenir ce type d'architecture seront définis. Ensuite, une méthode proposée pour la conception de l'architecture sera détaillée. Pour finir, nous résumerons les principaux avantages et inconvénients de cette méthode.

4.2 Prérequis

4.2.1 Principe de la méthode proposée

Le but est d'obtenir un partitionnement de la région reconfigurable du FPGA en *n* partitions PR qui pourront accueillir des tâches matérielles de tailles différentes. Pour cela nous avons introduit une couche supplémentaire sur la structure du FPGA (figure 4.1). La couche partitionnée permet de réaliser un partitionnement du FPGA et elle s'incère au-dessus de la couche «frame» (une frame est le plus petit élément reconfigurable d'un FPGA) comme le montre la figure 4.2. On obtient donc une organisation à trois couches, la première couche est fixe et elle dépend de la technologie, la seconde est conçue en fonction de l'application et en fonction du partitionnement minimal qui permet une efficacité maximale. La figure 4.3 montre un exemple de découpage, la première conception (a) représente un partitionnement de la région en trois partitions PR qui peuvent accueillir trois tâches matérielles (module PR) et une région est réservée pour la mise en mémoire de données sur des ressources de type BRAM. La seconde conception (b) montre le cas où une tâche matérielle a besoin de mémoire et de plus de ressources que ce que peut lui fournir une partition PR. Cette tâche est donc implémentée sur deux partitions PR et sur une partition de mémoire BRAM. La troisième conception (c) est utilisée dans le cas où la tâche matérielle a besoin de toutes les ressources de la région reconfigurable. Dans ce cas, le module PR utilise toutes les partitions PR de la région PR. Dans le flot de conception PBPR ainsi que dans les flots antérieurs (MBPR, DBPR et EAPR) la taille des régions reconfigurables est prédéterminée au début de la conception et la taille d'un module PR doit être inférieure ou égale à la taille de la région reconfigurable. Ces flots de conceptions ne permettent donc pas d'imbriquer ou de superposer les régions reconfigurables. De plus, ils ne permettent pas non plus de partitionner les régions reconfigurables en partitions partiellement reconfigurables. Ainsi, il est nécessaire d'étudier de nouvelles solutions pour surmonter ces restrictions qui ne permettent pas l'adaptation des partitions PR aux tâches matérielles de tailles différentes. Cette contrainte peut être levée grâce à une amélioration du flot de conception qui permet la création d'une nouvelle conception pour chaque type de partitionnement désiré. Avant de présenter cette amélioration du flot de conception, il est nécessaire d'expliquer les prérequis nécessaires à la

mise en place de ce flot de conception.

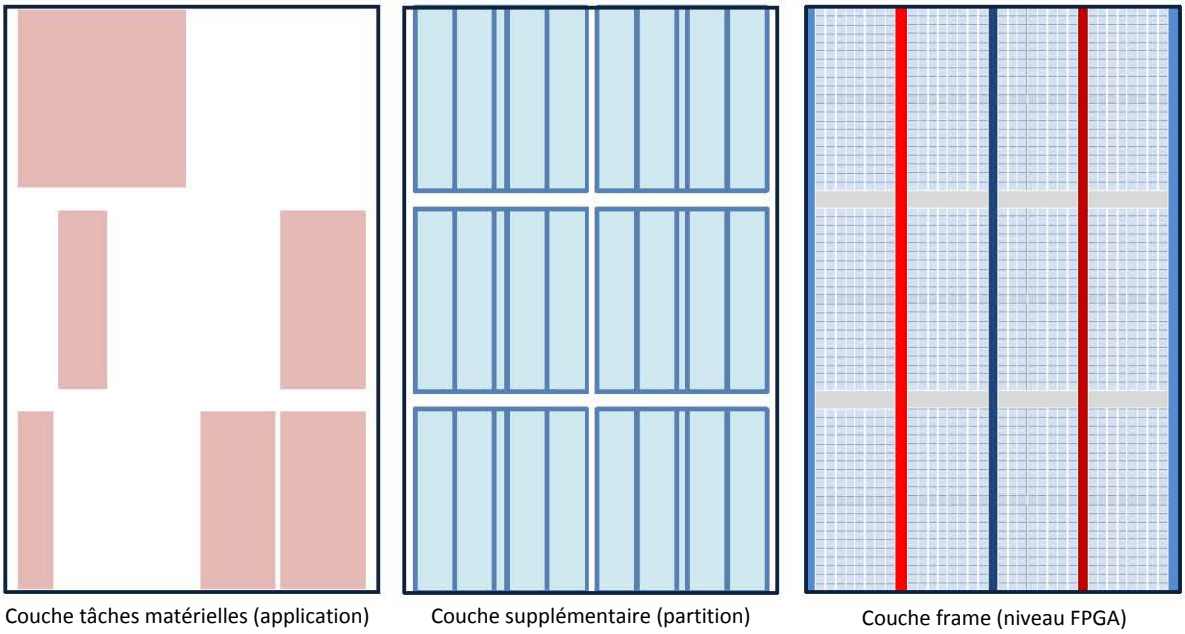


FIGURE 4.1 – Organisation des différentes couches pour une architecture adaptatives.

4.2.2 Architecture et conception pour les tâches matérielles de tailles différentes

Le principe de la méthode proposée consiste à partitionner la région reconfigurable en partitions PR pour implémenter des tâches matérielles de tailles différentes. Une architecture a été définie, elle est divisée en deux régions principales. La première région est statique (figure 4.4.a) et elle contient des tâches matérielles qui permettent la communication avec l'extérieur (USB, PCIe, Ethernet...), le contrôle des mémoires, le contrôle de la reconfiguration (processeur) et toutes les autres tâches matérielles statiques utiles au bon fonctionnement de l'application. La seconde région est reconfigurable (figure 4.4.b), elle peut être divisée en plusieurs partitions PR. Pour chaque partitionnement souhaités, une nouvelle conception de la région reconfigurable est créée avec des partitions PR de tailles différentes. La figure 4.4.b représente une conception avec une seule partition PR qui occupe toute la région reconfigurable, la figure 4.4.c représente une conception avec 4 partitions PR et la figure 4.4.d représente une conception avec 6 partitions PR. Avec cette méthode, on peut créer une conception pour chaque partitionnement souhaité

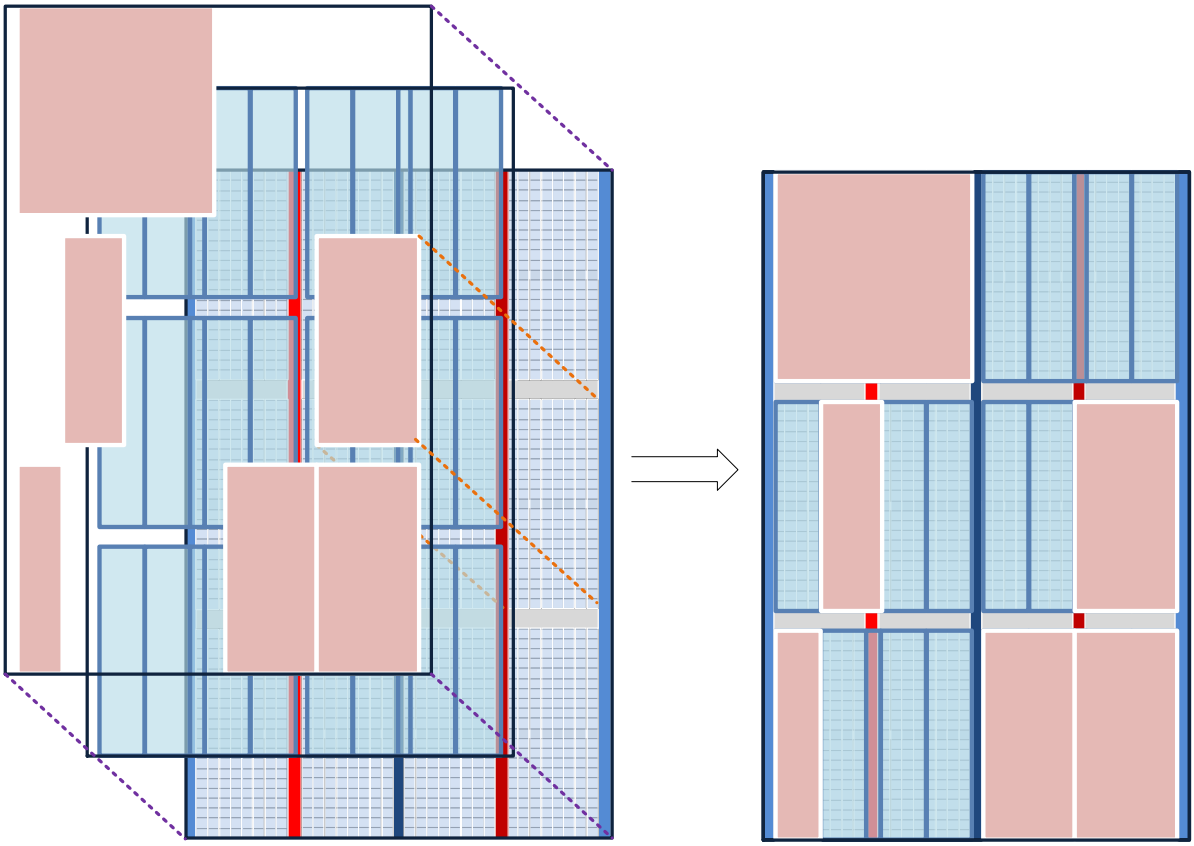


FIGURE 4.2 – Empilage des trois couches du FPGA.

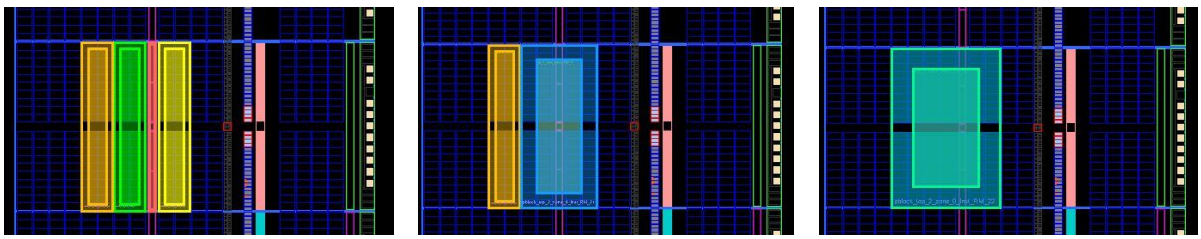


FIGURE 4.3 – Exemple de partitionnement de la région reconfigurable du FPGA en n partitions PR qui pourront accueillir des tâches matérielles de tailles différentes. (a) Sous-conception avec un partitionnement de la région en trois partitions PR, (b) sous-conception avec un partitionnement de la région en deux partitions PR, (c) sous-conception avec un région avec une seule partition PR.

de la région reconfigurable. L'architecture est donc composée d'une conception de base qui contient tous les éléments statiques du système et de plusieurs sous conceptions qui définissent

les différentes tailles des partitions PR de la région reconfigurable.

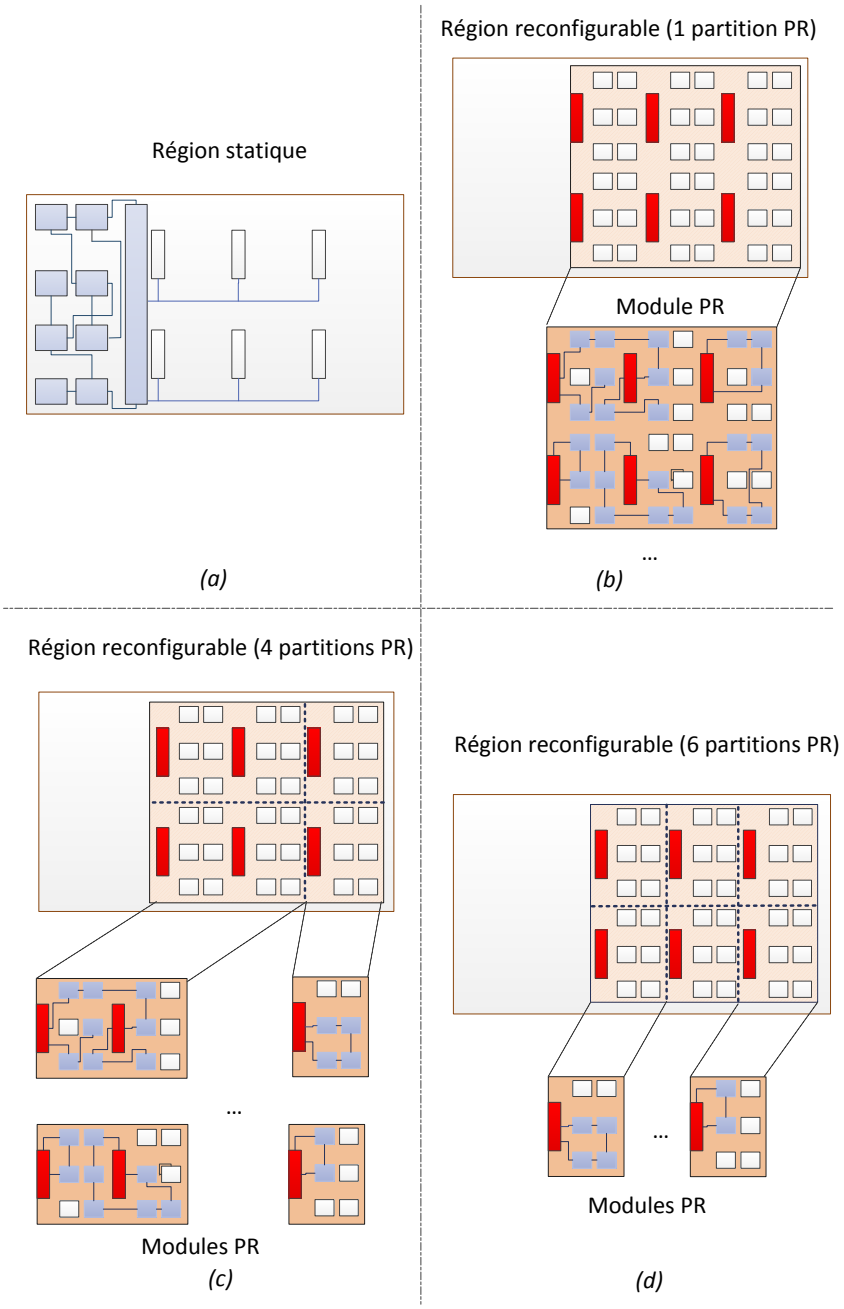


FIGURE 4.4 – Architecture permettant le placement de tâches matérielles (modules PR) de tailles différentes. (a) conception de base contenant les éléments statiques et le wrapper statique, (b) sous-conception de la région reconfigurable avec une partition PR, (c) sous-conception de la région reconfigurable avec quatre partitions PR, (d) sous-conception de la région reconfigurable avec six partitions PR.

4.2.3 Création d'un wrapper de connexion pour les tâches matérielles

Pour réaliser une architecture qui permet l'implémentation de tâches matérielles différentes sur des partitions PR, il est nécessaires de rendre interopérables les modules PR physiques sur des partitions, dont les entrées/sorties, sont figées. Il faut savoir qu'une région reconfigurable a un wrapper (entrées/sortie) figé, cette contrainte implique que les futures partitions PR auront aussi un wrapper fixe. Par conséquent les tâches matérielles implémentées sur ses partitions PR doivent avoir :

- le même nombre d'entrées et de sorties ;
- les entrées/sorties de même taille ;
- les entrées/sorties de même nom.

La figure 4.5 illustre les contraintes du wrapper. Les modules PR 1 et 2 sont entièrement compatibles avec la partition PR. Le module PR 3 a une entrée avec un nom différent, le module PR 4 a une sortie de taille différente et le module PR 5 a une sortie en plus. Toutes ces différences rendent incompatibles ces trois modules PR avec la zone reconfigurable.

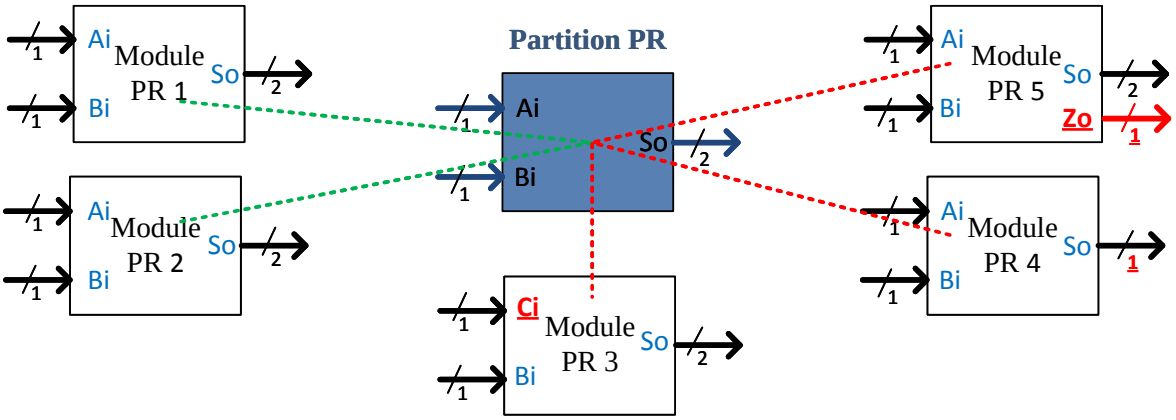


FIGURE 4.5 – Contraintes du wrapper d'une région reconfigurable ou d'une partition PR.

Pour résoudre ce problème d'architecture, nous proposons un wrapper générique composé d'une partie dynamique et d'une partie statique. Le Wrapper statique n'est pas reconfigurable, il est connecté à une région reconfigurable ou à une partition PR qu'il contrôle en prenant en charge la sauvegarde et la restauration du contexte entre deux reconfigurations. En plus de cette sauvegarde de contexte, une fonction (swap) permet de connaitre l'état du module PR (en

cours de traitement, traitement terminé). Cette fonction permet de savoir si l'ont peut ou non reconfigurer la partition PR avec une nouvelle tâche matérielle. Une mémoire de type FIFO est ajoutée au wrapper statique pour bufférisier les données d'entrées. Cette mémoire permet de ne pas perdre de données entre deux reconfigurations. Le wrapper statique assure aussi la communication entre la région reconfigurable et le reste du système.

Le wrapper dynamique est spécifique à chaque tâche matérielle et il fait partie du module PR qui est instancié sur la partition PR. Le but du wrapper dynamique est de simplifier et de généraliser la conception du wrapper statique qui sera unique pour toutes les partitions PR d'une même conception. Le wrapper dynamique inclut une interface qui lui permet de communiquer avec le wrapper statique. La complexité de l'interface dépend de la complexité et du nombre de données à transférer. Sur la figure 4.6, on peut voir l'exemple de deux wrappers dynamiques et du wrapper statique. Le wrapper statique est connecté à la partition PR par l'intermédiaire de différents signaux (Horloges, Reset, validations, informations, donnes...). Le wrapper dynamique permet d'harmoniser les entrées/sorties entre les différents modules PR. Sur la figure 4.6 le module PR A utilise deux bus bidirectionnels, deux signaux en entrées et trois de sorties (plus les signaux d'horloge, de reset et d'activation) ces signaux sont différents du module PR B. Le wrapper dynamique permet donc d'avoir en entrée et en sortie de la partition PR des signaux identiques sur les deux modules. Les signaux entre les deux wrappers peuvent être regroupés sous la forme d'un bus de données et d'un bus de contrôle.

La figure 4.7 montre comment fonctionne le wrapper dans le cas où la région reconfigurable et divisée en plusieurs partitions. Le wrapper statique doit être capable de gérer toutes les partitions PR de la conception, pour cela il doit avoir suffisamment de mémoire pour gérer les données de chaque partition PR. Les modules PR qui utilisent deux partitions PR sont connectés à un ou deux bus d'interface en fonction de leurs besoins en signaux d'entrées et de sorties. Sur la figure 4.7, le module PR B utilise deux partitions ainsi que les deux interfaces de connexion alors que le module PR C utilise qu'une seule interface de connexion.

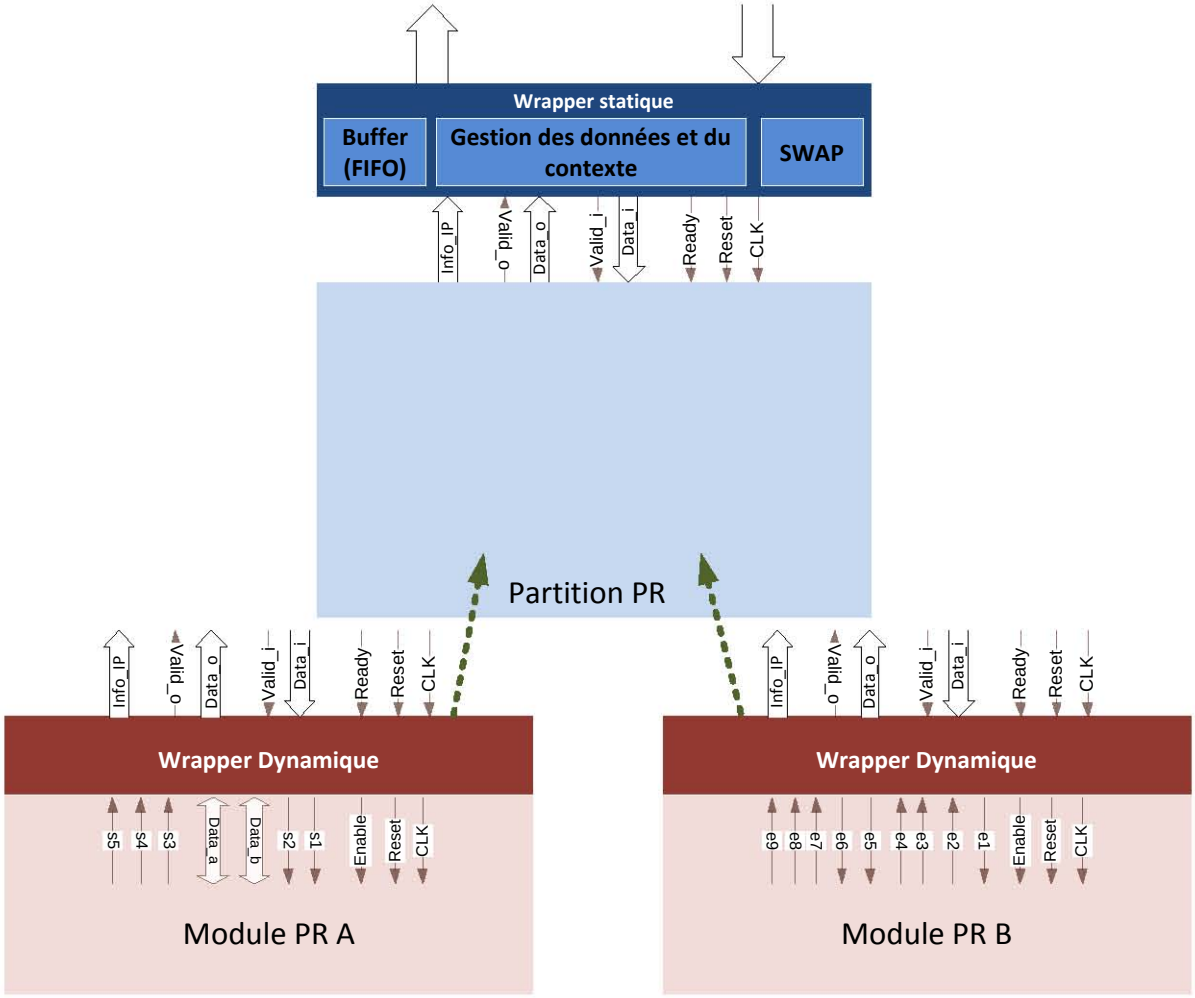


FIGURE 4.6 – Wrapper statique et dynamique d'une partition PR.

4.2.4 Gestion des interfaces de communications

Les systèmes dynamiques et partiellement reconfigurables sont mis en œuvre en séparant les ressources statiques des ressources reconfigurables. L'une des conditions essentielles est de garantir une connexion sécurisée entre la région statique et la région reconfigurable avant et après la reconfiguration. Les flots de conceptions antérieures proposent l'utilisation de buffer à trois états implémentés en dur dans l'architecture des FPGA Xilinx. Cette solution n'est plus utilisée depuis que les buffers à trois états ont été retirés des nouvelles architectures. L'alternative à cette solution est l'utilisation de macro préroutée aussi appelés bus macros. Par la suite la reconfiguration basée sur le flot de conception PBPR est apparue. Les bus macros sont remplacés par des

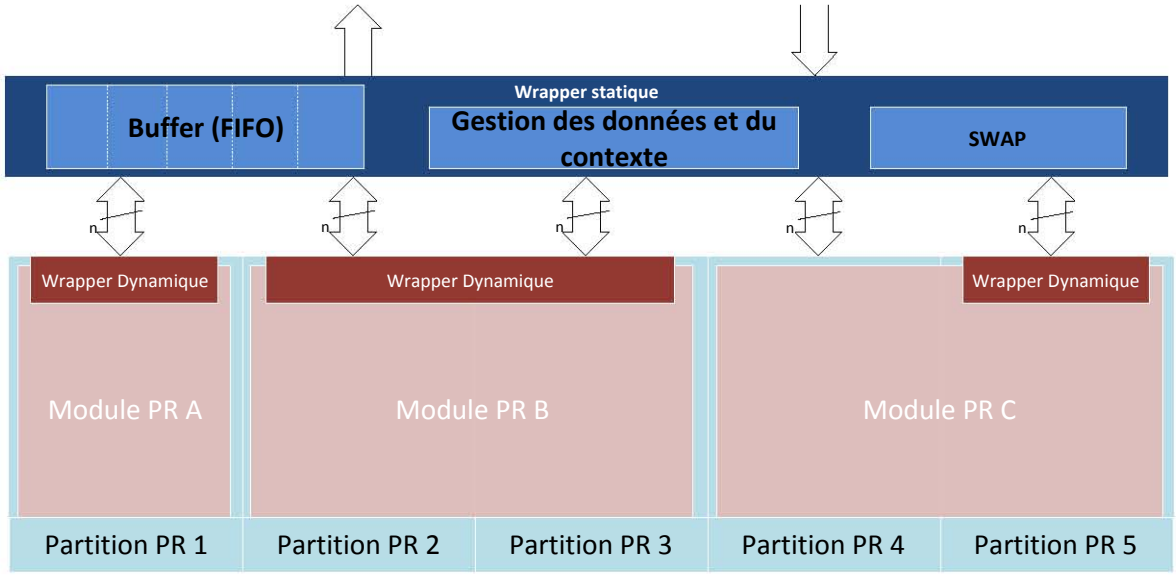


FIGURE 4.7 – Exemple avec plusieurs partitions PR.

pins partitions qui sont automatiquement placés dans la région reconfigurable. Cette interface de communication est mise en œuvre sur des blocs de type Look Up Tables (LUT). Les pins partitions sont caractérisées par leur type (entrée ou sortie), par leur placement (coordonnées XY de la SLICE où est instancié la LUT), par leur élément logique de base (BEL) et par leur point de connexion [VBR+96].

Pour que les connexions des pins partitions restent cohérentes entre les différentes conceptions, il faut que le placement de celle-ci reste identique entre les différentes conceptions. Les outils actuels n'offrent pas de solution pour exporter le placement complet des pins partition. Cela permet d'assurer la communication entre les différentes régions. Il est donc nécessaire de contraindre le placement des Pins Partition sur les conceptions additionnelles (conception avec un nouveau partitionnement de la région reconfigurable).

Les outils de conceptions actuels offrent la possibilité de contraindre la SLICE où est implémenté la pin partition en utilisant les coordonnées XY de celle-ci, ensuite il est possible de choisir l'élément logique de base qui sera utilisée (une des LUT disponibles dans la SLICE). Par contre, le routage de la pin partition sur l'entrée d'une LUT n'est pas contraignable avec les outils de conceptions actuels. La pin partition est automatiquement placée et routée sur l'une des entrées de la LUT.

À titre d'exemple, la figure 4.8.a montre la coupe d'une slice composée de 4 LUT6 (avec 6 entrées chacune - Virtex 5). La figure 4.8.b montre le placement d'une pin partition pour une première conception. La pin Partition est placée sur la LUT6 D de la slice et elle est routée sur l'entrée D6 de la LUT6. Cette connexion est compatible avec la région statique. La figure 4.8.c montre la mise en place d'une Pin Partition sur une autre conception. Dans ce cas, la pin Partition est contrainte pour être placée sur la même Slice que précédemment et pour utiliser la même LUT6 D. Par contre, vu qu'il est impossible de contraindre le routage, il est possible que l'outil choisisse une entrée différente que pour la conception précédente. Dans l'exemple de la figure 4.8.b.c, nous pouvons voir clairement que le positionnement de la pin partition d'entrée est différente d'une conception à l'autre. Cette différence de connexions entre les deux conceptions (figure 4.8.b.c) ne permet pas d'établir une connexion entre les partitions PR et la région statique.

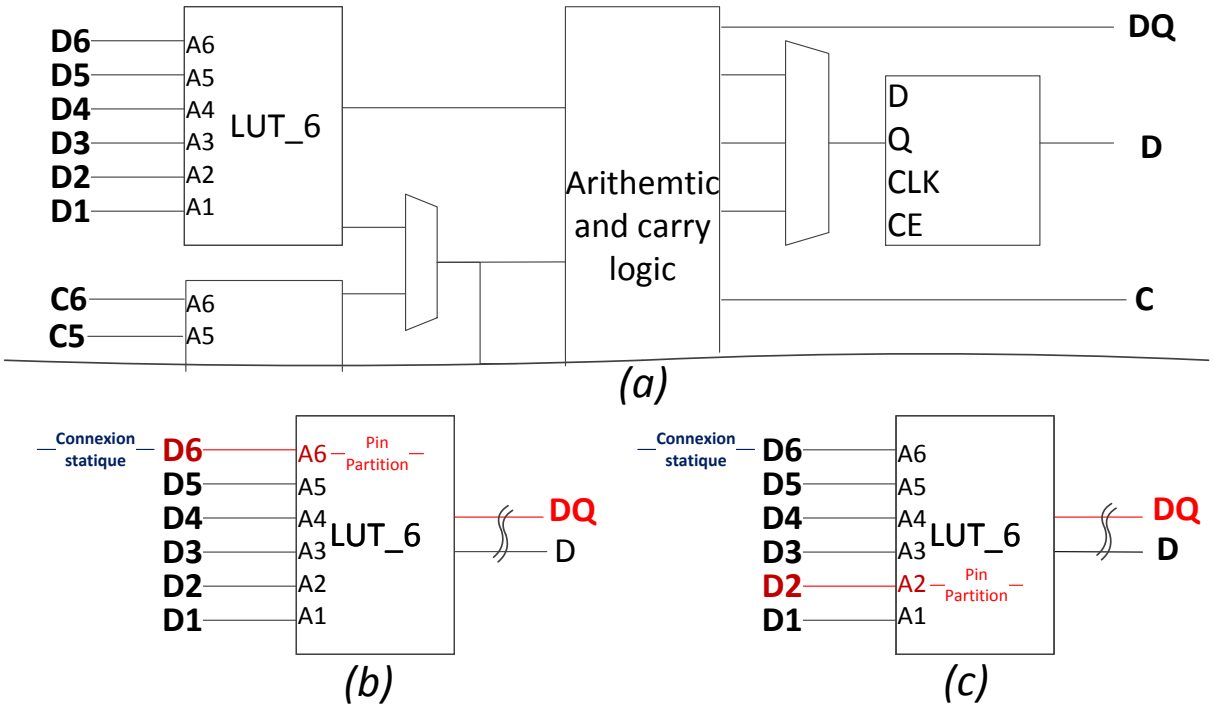


FIGURE 4.8 – Placement d'une pin partition. (a) Schéma simplifié d'une SLICE d'un FPGA Xilinx Virtex-5. (b) et (c) : Exemple de placement d'une pin partition

4.3 Nouveau flot de conception : Variable Partition Based Partial Reconfiguration (VPBPR)

4.3.1 Introduction

Nous avons vu dans la partie précédente les prérequis nécessaires pour réaliser une architecture reconfigurable qui peut accueillir des tâches matérielles de tailles différentes. Pour réaliser ce type d'architectures, il est nécessaire de modifier les flots de conceptions actuelles qui ne permettent pas le placement des tâches matérielles de tailles différentes. Dans la suite de cette partie, nous allons présenter une modification du flot de conception utilisé par Xilinx pour rendre possible le placement des tâches matérielles de tailles différentes sur une architecture reconfigurable. Le flot de conception peut être divisé en deux parties, la première partie permet de définir et de décrire (Netlist/VHDL) les différents éléments de l'architecture et la seconde partie permet de concevoir l'architecture globale qui permet le placement des tâches matérielles de tailles différentes sur une région reconfigurable.

4.3.2 Présentation du flot

4.3.2.1 Définitions des éléments de l'architecture

Cette première partie du flot de conception permet de définir et de décrire les éléments de l'architecture qui nous permettent de répondre aux prérequis définis dans le début de ce chapitre. La figure 4.9 illustre les différentes étapes qui permettent la définition des éléments de l'architecture. Ces définitions sont des descriptions VHDL ou des Netlist que l'on peut réaliser sous ISE (Integrated Software Environment) [Xilc] qui est un environnement de développement intégré aux outils Xilinx.

La première étape est la définition de tous les éléments qui constitueront la région statique de l'architecture. Ces éléments forment un système minimum et ils sont la base de notre architecture qui peut être composée d'éléments de communication avec l'extérieur (PCIe, Ethernet, USB, DVI...), de contrôleur mémoire, de contrôleur de reconfiguration, d'un processeur (software) et d'autres tâches matérielles statiques. Une fois tous les éléments décrits, il faut les

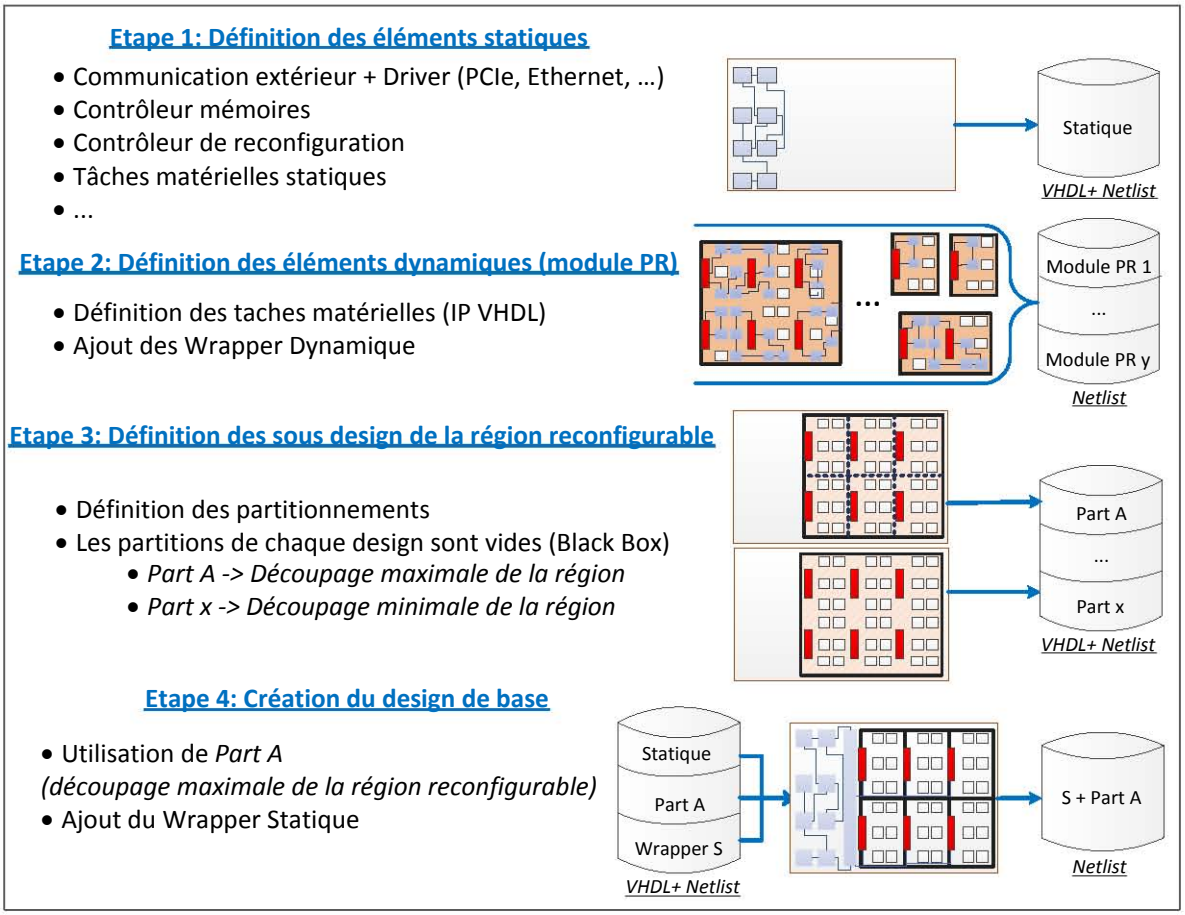


FIGURE 4.9 – Hiérarchie d'un projet contenant deux partitionnements de la région reconfigurable. Le premier contient une partition PR et le second en contient 6.

rassembler pour créer l'architecture statique du projet.

La seconde étape est la définition de tous les éléments dynamiques (modules PR) qui seront implémentés sur des partitions PR. Dans cette étape les modules PR sont des nouvelles tâches matérielles qui sont créées spécifiquement pour ce projet ou se sont des IP. Ensuite, il faut ajouter aux tâches matérielles ou aux IP un wrapper dynamique. L'ajout du wrapper rend les modules PR interopérables avec les partitions PR. Les nouvelles descriptions VHDL sont ensuite synthétisées pour estimer leurs tailles et les ressources qu'elles occuperont sur le FPGA et pour créer un fichier de description par module PR appelé « Netlist ».

La troisième étape est la définition des sous-conceptions qui contiennent les partitionnements de la région reconfigurable. En fonctions des ressources qu'utilisent les modules PR, on

peut définir la dimension des partitions PR nécessaire à l'implémentation des tâches matérielles de tailles différentes. Chaque partitionnement de la région reconfigurable sera réalisés dans un fichier VHDL qui contient un ou plusieurs composants qui représentent chaque partition PR. Chaque partition PR est un composant vide, c'est une boite noire (Black Box) qui n'est définie que par ses ports d'entrée et de sortie. Dans l'exemple sur la figure 4.9, le projet contient deux partitionnements de la région reconfigurable. Le premier contient une partition PR et le second en contient six. Vous trouverez en annexe 2 et 3 un exemple pour les descriptions VHDL de ces deux partitionnements.

La quatrième étape est la création de la conception de base de l'architecture. Cette conception regroupe la définition des éléments statiques de l'étape 1 et la sous-conception avec un partitionnement maximal de la région reconfigurable en partition PR (étape 3 – Part A). Pour permettre la communication entre la région reconfigurable et la région statique de l'architecture, le wrapper statique est ajouté.

Cette première partie du flot de conception nous a permis de définir et de décrire les éléments qui nous seront utiles dans la conception de l'architecture. De plus, pendant ce flot, il nous est possible de tester les différentes configurations avec des TestBench pour valider les descriptions VHDL (simulation sous Modelsim).

4.3.2.2 Conception de l'architecture

La seconde partie du flot de conception permet de concevoir l'architecture globale qui permet le placement des tâches matérielles de tailles différentes sur une région reconfigurable. La figure 4.10 illustre le flot de conception proposé qui s'appuie sur le flot de conception PBPR (partitions based partial reconfiguration) de Xilinx. Notre nouveau flot est décomposé en cinq étapes principales. Dans ce flot, la conception de base (Conception A) correspond à la définition des régions statique et reconfigurable. Dans cet exemple, le partitionnement initial de la conception A est composé de six partitions PR. Dans la conception B deux partitions PR sont fusionnées afin de pouvoir implémenter des modules PR qui nécessitent plus de ressources. La conception C est un autre exemple avec une seule partition PR qui recouvre toute la région re-

configurable. Dans les conceptions B et C, seules les partitions PR de la région reconfigurable sont définies en utilisant des informations importées de la conception A. Cette opération peut être répétée en fonction du nombre de sous conceptions requis. Toutes les étapes du flot de conception peuvent être réalisées avec l'outil PlanAhead ou en ligne de commandes dans un shell (inviter de commande) Xilinx.

Étape 1 : Mise en œuvre de la conception initiale

Le fichier netlist de la région statique et de la sous-conception de la région reconfigurable avec le plus petit partitionnement est importé dans PlanAhead. Dans la conception initiale la région reconfigurable est définie pour que ses partitions PR permettent un placement efficaces des tâches matérielles (ou IP) pour qu'elles utilisent le moins de ressources. À partir de cette partition, une description native du circuit (netlist) est générée. Cette netlist correspond à la mise en œuvre de la conception initiale qui contient la partie statique et des partitions PR qui sont vides (Black Box). Les interfaces de communication (pins partitions) entre la région statique et les partitions PR sont incluses dans la partie statique et elles sont placées automatiquement. Cette étape est semblable au flot PBPR standard de Xilinx, les étapes suivantes sont ajoutées au flot pour rendre possible le placement des tâches matérielles de tailles différentes sur des partitions PR.

Étape 2 : Extraction des informations sur les Pin Partition.

Pour chaque nouvelle sous conception, il est nécessaire d'importer des informations sur les interfaces de communication (Pin Partition) entre la région statique et la région reconfigurable de la conception initiale. Ces informations sont extraites du fichier NCD de la conception initial généré à l'étape 1. Ces informations rassemblent tous les renseignements nécessaires aux placements et aux routages des pins partitions, elles seront utilisées pour toutes les futures sous conception. Les informations sur l'emplacement de chaque pin partition regroupent son type d'élément logique (BEL) et les coordonnées de la SLICE qu'elle utilise. Un exemple d'extraction est donné en annexe 4. Les informations de routage sur les pins partitions doivent aussi être préservées pour chaque nouvelle conception. Elles se trouvent dans le fichier NCD

de la conception initial. Ce fichier est entièrement crypté. Il faut donc utiliser la commande NCD2XDL pour décrypter ce fichier à l'aide d'un langage de conception. XDL (Xilinx Design Language) [BKT11] est un langage permettant de récupérer les informations sur le routage des pins partitions d'une conception (voir extrait XDL annexe).Un exemple d'extraction est donné en annexe 5.

Étape 3 : Création d'un sous design.

Dans cette étape, une sous conception de la région reconfigurable est créée pour la mise en œuvre d'un nouveau partitionnement. Il est important de noter que la partie statique de la conception n'est pas utilisée dans cette étape, cela permet d'alléger la charge de travail et de réduire le nombre de fichiers à manipuler. Cependant, le fichier de contrainte du design initial est appliqué pour contraindre le placement des pins partions. Cette importation permettra de réserver les slices et les éléments logiques (BEL) et de s'assurer qu'elles seront utilisées seulement pour l'implémentation des pins partitions.

Étape 4 : Reroutage des pins partitions.

Un fichier XDL est généré à partir du fichier NCD de la nouvelle conception grâce à la commande NCD2XDL. Ce fichier nous permet de connaitre le routage complet de la nouvelle conception. Ce fichier doit être modifié à l'aide des informations de routage extraites pendant l'étape 2. Les modifications apportées permettront de rerouter les pins partitions pour que leurs connexions soient communes entre toutes les conceptions. Enfin un nouveau fichier NCD est généré à partir du fichier XDL modifié et grâce à la commande XDL2NCD.

Étape 5 : Génération des bitstreams.

Dans cette étape, le bitstream de la partie statique et des bitstreams partiels vides pour chaque partition PR sont générés à partir de la conception initiale. Les bitstreams partiels vides (blank) permettent d'effacer la configuration d'une partition PR . Les bitstreams partiels de chaque Module PR sont générés à partir des sous-conceptions.

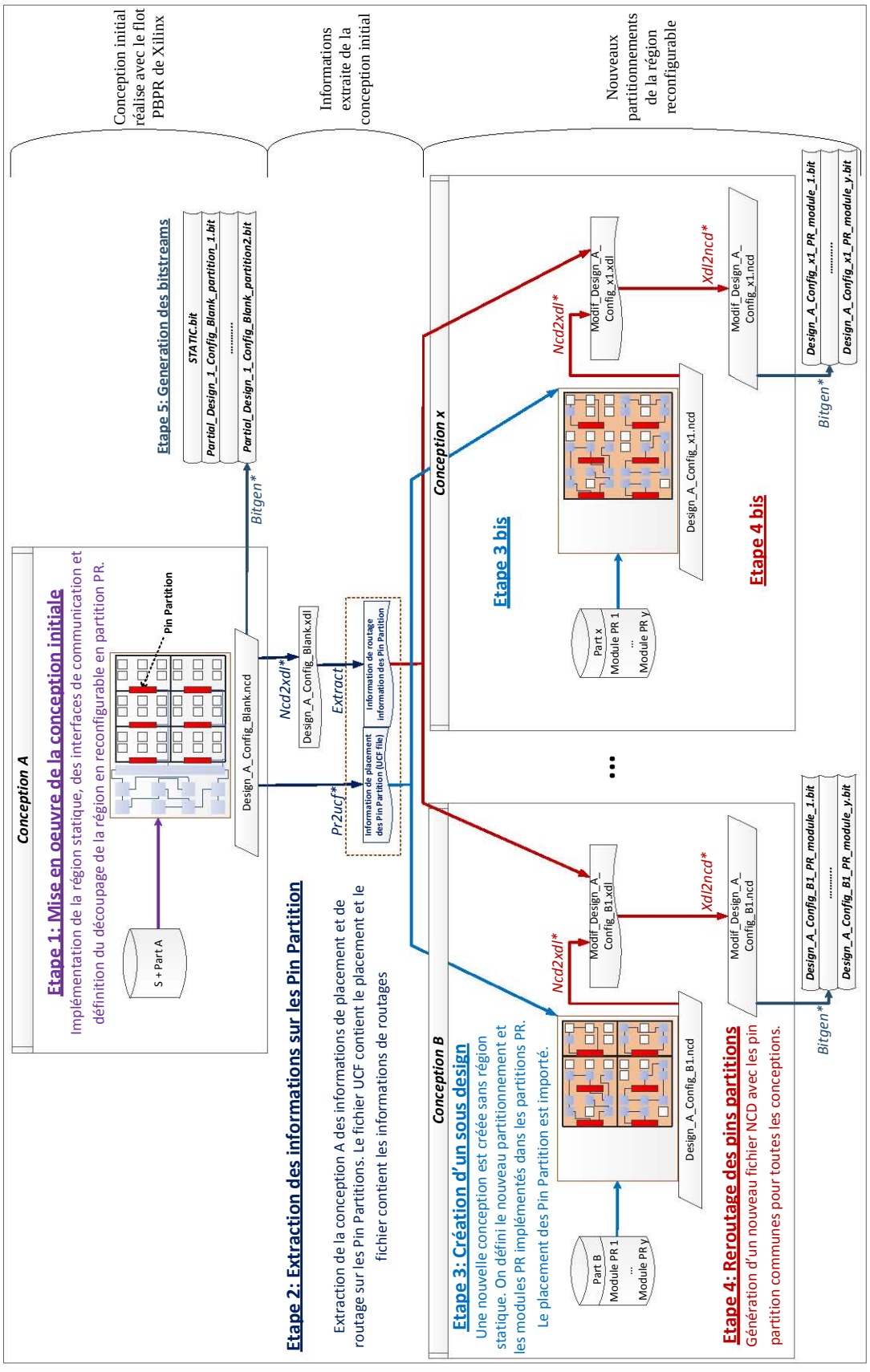


FIGURE 4.10 – Flot de conception pour la conception d'une architecture avec des partitions PR adaptable. *Fonction native de Xilinx.

4.3.3 Discussion

Le tableau 4.1 montre une comparaison des différents travaux sur l'adaptation des partitions PR aux tâches matérielles de tailles différentes.

TABLE 4.1 – Comparaison avec les différentes méthodes sur le principe de l'adaptation des partition PR (Xilinx PR communication :*ancienne/**actuelle)

Référence	Adaptation des partitions PR	Méthode	Cible FPGA	PR communication
Walder (2004) [WP04, WSP03]	Non (proposition du principe)	bus de communication spécifique (Task Communication Bus)	Virtex 800	Bus-macro*
Hubner (2006) [HB06]	Non (manipulation de bitstream)	Macro de routage. Manipulation de bitstream (Jbits)	Virtex 2	Bus-macro*
Hagemeyer (2007) [HKKP07a, HKKP07b]	Non (BUS pour l'adaptation des partition PR)	Nouveau flot de conception (INDRA). BUS spécifique	Virtex 2	Bus-macro*
Pionteck (2009) [PAKM09, PKAM09, PKA06]	Oui (Hard Macro)	Architecture NoC. Xilinx (PR flot) modifié. Ajout d'une Hard macro	Virtex 4	Bus-macro*
Oetken (2010) [OWTK10]	Oui (ReCoBus)	Toolkit spécifique(ReCoBus et I/0 bars)	Virtex 2	Bus-macro*
Méthode proposée	Oui(fichier de contrainte)	Utilisation du PR flot et des commandes proposées par Xilinx	Virtex 4, 5 et 6	Pin-Partition**

Le premier constat que l'on peut faire, est que toutes ces méthodes utilisent soit une modification du flot de conception avec l'ajout de nouveaux logiciels ou soit l'ajout de modules hardware à l'architecture. On remarque aussi qu'aucune des méthodes n'est compatible avec les dernières technologies telles que les FPGA Virtex 5 et 6 et leur pin partition qui permettent la communication avec les régions reconfigurables. Walder [WP04, WSP03], Hubner [HB06] et Hagemeyer [HKKP07a, HKKP07b] proposent de nouveaux principes pour la communication des partitions PR dans le cas d'une adaptation des partitions PR . Tous leurs principes sont basés sur l'ajout d'un bus spécifique ou de macro de routage. De plus, ses techniques nécessitent soit un nouveau flot de conception ou une manipulation des bitstreams de reconfigurations. Ces méthodes ont permis de définir les principes de l'adaptation des partitions PR aux tâches matérielles de taille différente. Pionteck [PAKM09, PKAM09, PKA06] et Oetken [OWTK10] ont proposé des techniques qui permettent de réaliser cette adaptation. Leurs méthodes est basée sur le flot de conception de Xilinx qui a été modifié pour permettre l'ajout d'un système de communication. Oetken utilise un bus de communication spécifique le ReCoBus alors que Piontek utilise une hard macro pour réimplémenter la partie statique de la conception et les bus-macro de communication pour chaque nouvelle conception avec un partitionnement différent. Cette méthode oblige le concepteur à réimplémenter l'ensemble du projet pour chaque nouvelle conception, il oblige aussi à créer une nouvelle hard macro pour chaque projet.

Les principaux avantages de notre méthode sont qu'elle est entièrement compatible avec le flot de conception de Xilinx et qu'elle ne nécessite pas l'ajout d'une surcouche logiciel ou matériel (Hard macro ou Bus spécifique). De plus, en choisissant de contraindre nos sous-conceptions nous n'avons pas besoin de reimplementer la partie statique de notre conception pour chaque nouveau partitionnement de la région reconfigurable. Le dernier avantage est que notre conception initiale et son fichier de contrainte ne sont générés qu'une seule fois et ensuite on importe seulement les contraintes pour toutes les autres sous-conceptions.

La solution proposée augmente la flexibilité des FPGA qui utilisent la reconfiguration dynamique sans complexifier le flot de conception. Pour le moment, l'extraction du routage se fait manuellement, pour rendre ce flot de conception plus simple d'utilisation il sera nécessaire d'automatiser cette partie.

4.4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode qui permet le placement des tâches matérielles de tailles différentes sur des partitions PR, pour comprendre le fonctionnement de cette méthode nous avons présenté les prérequis nécessaires à la conception d'une architecture adaptative, il est nécessaire de prendre en compte les contraintes de la reconfiguration dynamique. Tout d'abord, une région partiellement reconfigurable (région PR) est définie lors de la création du circuit sachant que sa taille, sa forme et sa position ne peuvent pas être modifiées après son implémentation sur le FPGA. Une seconde contrainte est présente au niveau de la communication d'une partition PR avec le reste du système (statique ou autre partition PR) qui est réalisée par l'intermédiaire d'interfaces, cette interface est plus communément appelée « wrapper ». Nous retiendrons la nécessité d'utiliser un wrapper de connexion générique pour chaque tâche matérielle qui permet leurs interopérabilités avec les partitions PR.

La contribution principale de ce chapitre est d'apporter au concepteur une méthode facile à utiliser pour adapter les partitions PR aux modules PR de tailles différentes, cette méthode s'appuie sur le flot de conception de Xilinx et sur de nouvelles techniques pour contraindre le placement des pins partitions. Cette méthode permet aussi d'éviter l'importation de la conception statique pour chaque nouvelle conception avec un partitionnement différent, et permet ainsi une réduction de la taille importante des fichiers à manipuler. Pour finir, la méthode et l'architecture proposées permettent d'augmenter l'efficacité des régions reconfigurables partitionnées en partitions PR.

Chapitre 5

Étude et validation de la méthode

Sommaire

5.1 Introduction

Dans le chapitre précédent, nous avons proposé une nouvelle méthode qui permet le placement des tâches matérielles de tailles différentes sur des partitions PR. Dans ce chapitre, nous allons évaluer notre méthode sur une application spécifique à un traitement vidéo. Dans un premier temps nous présenterons deux architectures potentielles pour l'implémentation des tâches matérielles de tailles différentes sur des partitions PR. Pour finir, nous présenterons une étude sur le gain d'efficacité en surface en fonction de plusieurs applications. Pour chaque application, nous étudierons les partitionnements possibles de la région reconfigurable. Cette étude nous permettra d'évaluer l'impact du partitionnement sur l'efficacité en surface.

5.2 Méthode d'évaluation

Dans l'introduction de cette thèse nous avons défini une métrique qui nous permet de quantifier l'efficacité en surface pour le placement des tâches de tailles différentes sur une région reconfigurable. Cette métrique sera aussi utilisée pour rechercher le partitionnement optimal pour une application donnée ou pour une classe d'applications. Dans le cas d'une application qui utilise une architecture avec partitionnement, nous définissons les termes suivants :

- TMi : Tâche matérielle i ;
- A : Application composée d'un ensemble de tâches matérielles de tailles différentes (équation 5.1) :
- TTM : Taille d'une tâche matérielle ;
- TRR : Taille de la partition ou des partitions reconfigurables occupées par la tâche matérielle TM ;
- Es : Efficacité en surface (équation 5.2).

$$A = \{TM1; TM2; \ldots; TMn\} \tag{5.1}$$

$$ES = \frac{TTM}{TRR} \tag{5.2}$$

5.3 Architectures potentielles

La méthode proposée pour le placement des tâches matérielles de tailles différentes sur des partitions PR peut être validée sur différents types d'architectures de FPGA. On peut distinguer deux classes principales. La première est basée sur une architecture de type bus et la seconde est de type NOC. Dans ce travail, l'évaluation a été faite sur l'architecture de type BUS sur une plateforme FPGA de type Xilinx Virtex 5.

5.3.1 Architecture de type bus

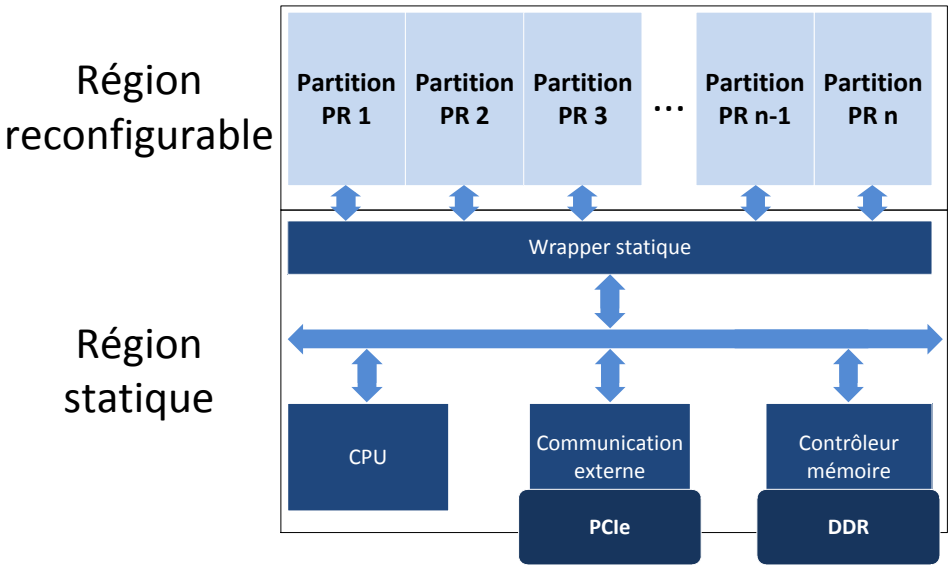


FIGURE 5.1 – Architecture reconfigurable partitionnée de type BUS.

La figure 5.1 illustre une des deux architectures cibles possibles, à base de mémoire partagée accessible par un bus. La région reconfigurable peut être découpée en partitions PR grâce au flot de conception présenté dans le chapitre précédent. Dans ce cas, nous considérons des tâches pouvant chacune être implémentées sur une ou plusieurs partitions PR de la région reconfigurable. La figure 5.2 illustre plusieurs exemples de configurations avec une région reconfigurable non partitionnée et avec une région reconfigurable partitionnée. L'architecture a été pensée initialement pour faire communiquer les différentes tâches par le biais d'une mémoire partagée sur un bus. Dans ce cas, la région reconfigurable doit pouvoir accéder au bus. Pour cela, la région

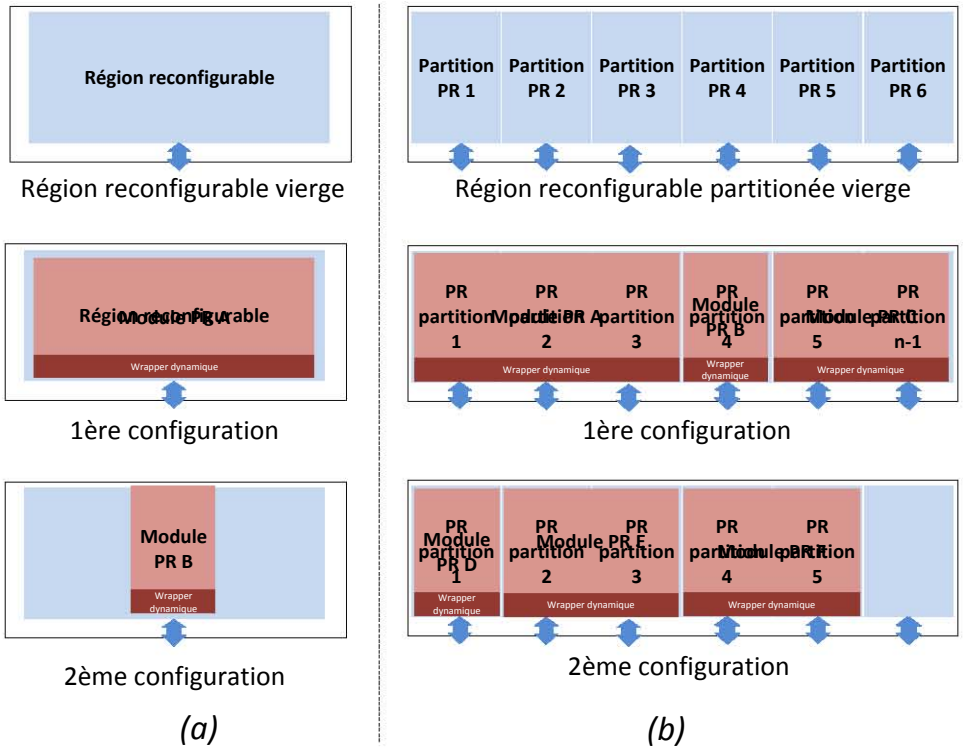


FIGURE 5.2 – Exemple d'implémentation d'une architecture reconfigurable de type BUS. (a) Région reconfigurable non partitionnée (b) Région reconfigurable partitionnée.

reconfigurable est connectée à un wrapper statique et les partitions PR ont chacune un wrapper dynamique. Notons qu'à l'exception de la région reconfigurable, le reste de l'architecture est flexible et peut être adapté aux besoins de l'application.

5.3.2 QNoC

La figure 5.3 illustre une autre architecture cible possible de type QNoC [Jov09] développée dans notre équipe de recherche. Le réseau QNoC hérite de la plupart des propriétés d'un réseau CuNoC. Un réseau CuNoC est caractérisé par un taux faible de ressources nécessaires pour son implantation, par une connexion spécifique entre ses routeurs et les tâches associées au réseau et par un algorithme de routage permettant l'acheminement des paquets et des messages même dans des conditions de reconfiguration dynamique du réseau. Cependant, un réseau CuNoC montre des résultats médiocres pour des conditions de communication ou de transfert de paquets nécessitant une bande passante élevée entre plusieurs tâches. Dans le but de réduire

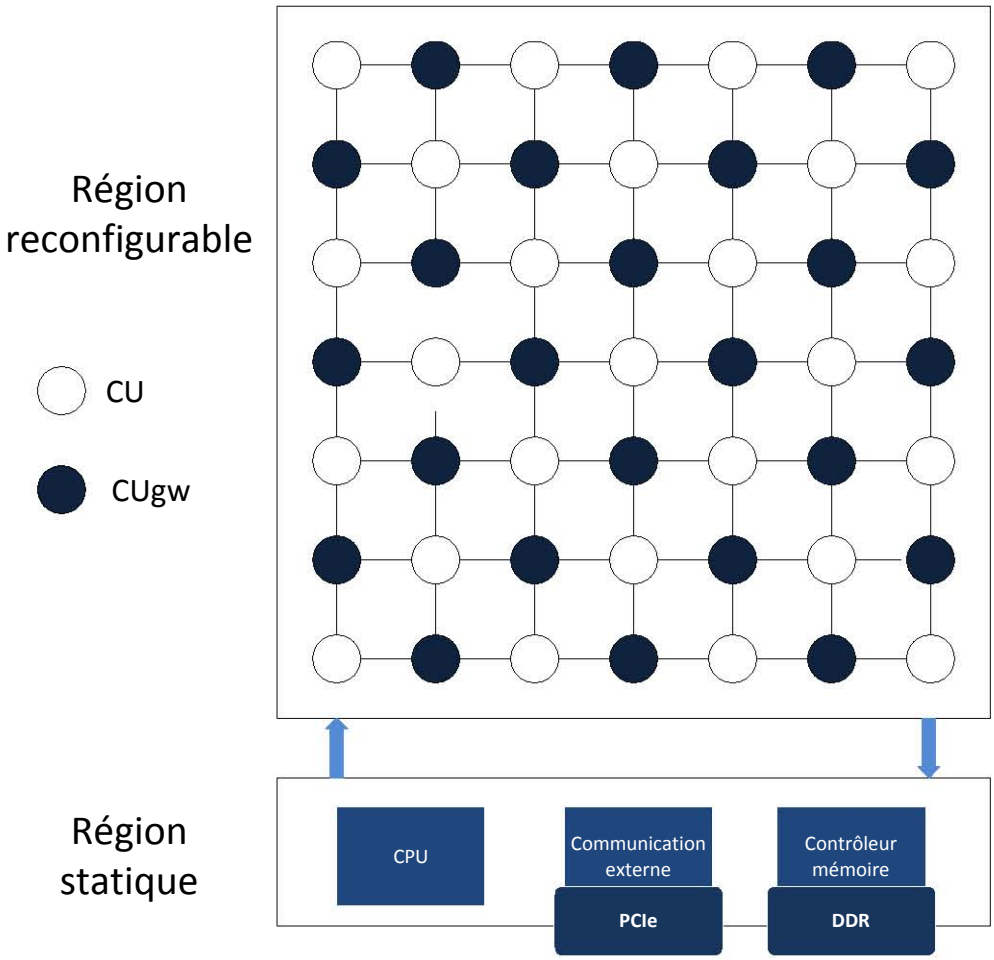


FIGURE 5.3 – Architecture reconfigurable partitionnée de type QNoC.

ces inconvénients, des améliorations de l'architecture d'un réseau CuNoC ont été proposées et ont permis de développer une approche améliorée d'un réseau sur puce pour un système reconfigurable baptisé QNoC [JTW08]. Un réseau QNoC a également hérité à la fois des aspects du réseau CuNoC permettant le placement dynamique de tâches et de la procédure de construction d'un réseau d'une taille précise. La figure 5.3 illustre une architecture reconfigurable de type QNoC. Nous distinguons deux types de routeurs (CU - Unité de communication) : un routeur CU « classique » (classic CU) et un routeur CU « qui-cède-le-passage » (to-give-way CU - CUgw) [JTW07, JTWB07, JTWB09]. Entre ces deux types de CU, il n'y pas de différences dans les conditions normales de communication dans le réseau. C'est-à-dire, dans les conditions où des situations d'embouteillage n'apparaissent pas au sein du réseau. La différence entre ces deux types de routeurs se situe au niveau de leurs modules de contrôles. Les graphes flot de

contrôle de chacun de ces deux CU sont présentés la thèse de Slavisha Jovanovic [Jov09].

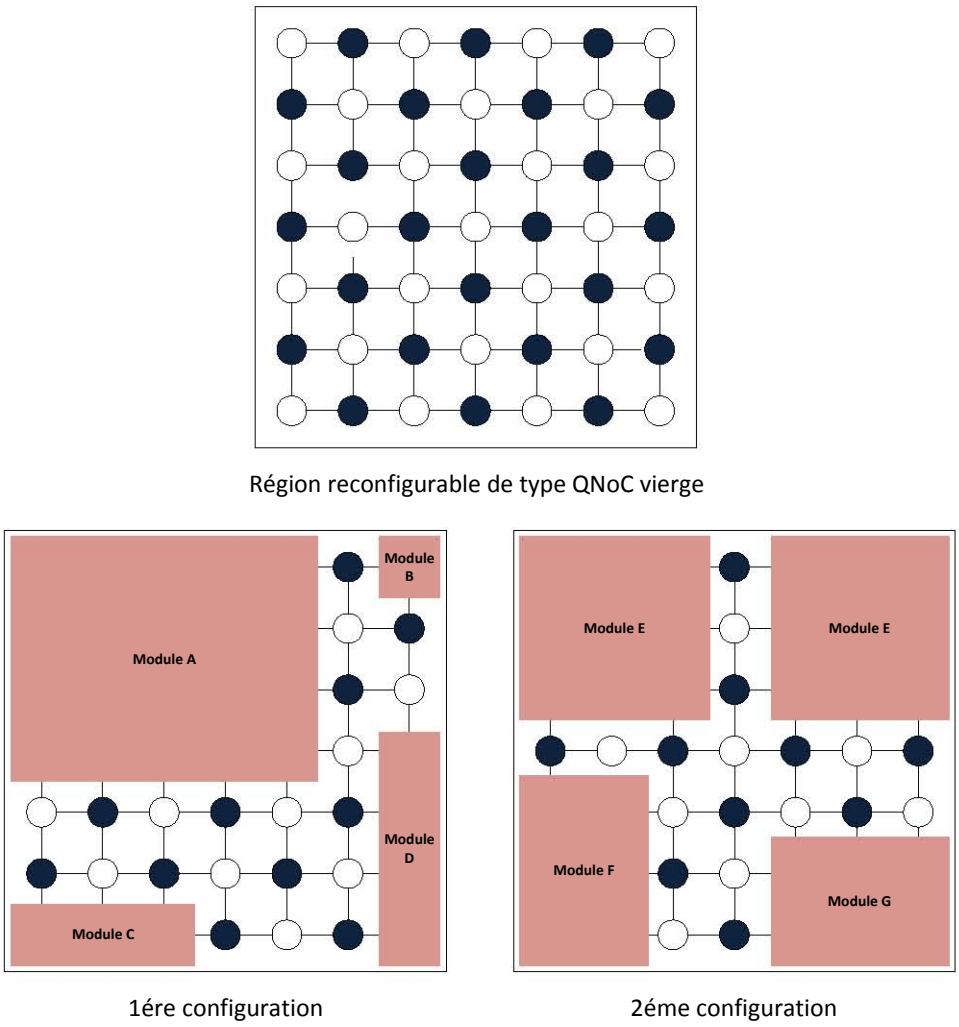


FIGURE 5.4 – Exemple d'implémentation sur une architecture QNoC.

Les autres éléments de l'architecture reconfigurable QNoC sont les modules qui correspondent aux tâches matérielles pouvant être implémentées sur la région reconfigurable. Notons que comme pour l'architecture précédente, à l'exception de la région reconfigurable, le reste de l'architecture est flexible et peut être adapté aux besoins de l'application. La figure 5.4 illustre un exemple d'implémentation sur une architecture de type QNoC.

5.4 Étude et validation globale

5.4.1 Application

5.4.1.1 Le transcodage vidéo

Pour valider notre architecture et notre flot de conception, nous avons utilisé l'application de transcodage vidéo traitée dans le cadre du projet ARDMAHN. Avec la diversité des normes de compression vidéo et des terminaux, les plateformes de transcodage ont besoin d'une solution architecturale flexible et efficace. Les décodeurs et les encodeurs vidéo de ces plateformes utilisent des techniques de codage entropique qui permettent de compresser et de décompresser des flux vidéo. Les techniques de codage entropique sont différentes d'un standard à l'autre, par exemple, en MPEG-2 un codage à longueur variable (VLC) est utilisé, alors que dans les normes de codage H.264 c'est un codeur à longueur variable avec un contexte adaptatif (CAVLC) ou un codeur arithmétiques binaires avec un contexte adaptatif (CABAC) qui peuvent être utilisés.

La présence simultanée de tous ces codeurs et décodeurs entropiques dans une puce entraînera des coûts importants en surface silicium et augmentera la consommation d'énergie [YCC01]. Plusieurs travaux ont été réalisés sur ce sujet pour proposer des solutions adaptatives [BIR10, Shi09]. Beaucoup de solutions sont basées sur des architectures logicielles qui utilisent un FPGA comme accélérateur matériel pour réaliser le codage ou le décodage entropique [YCC01, Tsa09]]. Les différentes solutions existantes visent à apporter une flexibilité par la mise en œuvre simultanée des codeurs ou des décodeurs entropiques, ou en utilisant la reconfiguration matérielle. Le gain obtenu en termes de superficie et de la flexibilité ne demeure pas significatif en raison de la complexité et de l'irrégularité de la taille de ces tâches matérielle [Shi09].

5.4.1.2 Tâches matérielles VLC et CAVLC

Nous avons choisi de valider notre architecture et notre flot de conception sur l'implémentation d'un accélérateur matérielle utilisé pour réaliser un codage entropique. Cet accélérateur matériel devra soit réaliser un codage pour des vidéos de type MPEG-2 ou H.264, il doit donc supporter deux tâches matérielles VLC et CAVLC. Ces deux décodeurs entropiques seront utilisés en fonction du flux vidéo désiré en sortie du système. La figure 5.5 montre les algorithmes

utilisés par CAVLC (figrure 5.5.a) et VLC (figrure 5.5.b). Ils reçoivent respectivement en entrées des blocs de coefficients 4x4 et 8x8 qui ont été au préalable transformés et quantifiés. En sortie, une fonction "toByte" permet de générer un flux vidéo compressé de tailles différentes appelé bitstream vidéo (figrure 5.5.c). La génération du bitstream vidéo se fait sous la forme d'octet, ce code de longueur variable doit être manipulé avec précaution pour éviter toute corruption dans le flux binaire généré.

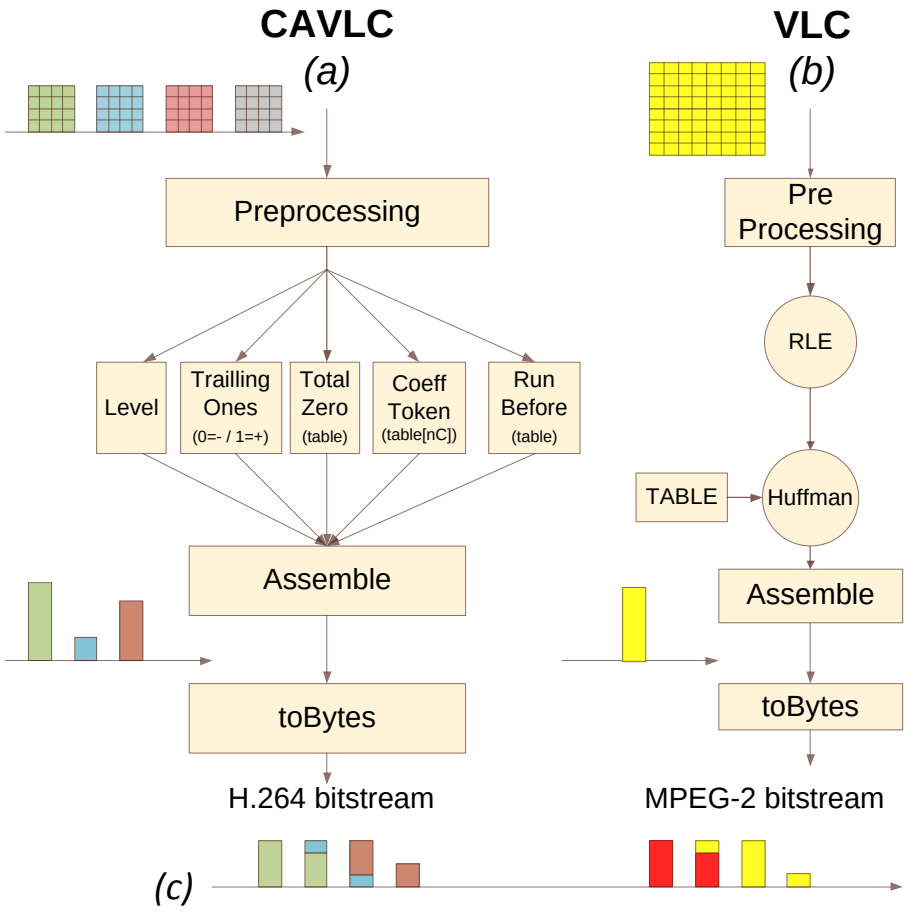


FIGURE 5.5 – Algorithmes utilisés par CAVLC (a) et VLC (b) et leur flux vidéo compressé de tailles différentes appelé bitstream (c).

Le tableau 5.1 résume les informations importantes sur les deux tâches matérielles. On remarque que ces deux tâches matérielles sont bien de tailles différentes, VLC occupe quasiment deux fois plus de ressources que CAVLC.

TABLE 5.1 – Information sur les codeurs entropiques CALVC et VLC

Codec	H.264	Mpeg-2
Encodeur	CAVLC	VLC
Entrée	Bloc $4x4$	Bloc $8x8$
Sortie	Bitstream H.264 de longueur variable	Bitstream Mpeg-2 de longueur variable
Fmax	160Mhz	150Mhz
Temps d'exécution par bloc	190ns	1,17μs
LUT	1190	2297
Slice	300	589
BRAM36	0	1

5.4.2 Encapsulation des tâches matérielles.

L'analyse des tâches matérielles CAVLC et VLC montre des différences et des similitudes en termes de traitement des données (bloc 4x4 pour CAVLC et 8x8 pour VLC). Des signaux de commande sont égalements utilisés en particulier pour la gestion des données entrantes et pour valider le flux binaire de sortie. Les similitudes entre les deux tâches sont exploitées pour créer le wrapper pour encapsuler les deux tâches matérielles (figure 5.6). Le wrapper statique (figrure 5.6.a) est constitué d'une mémoire capable de manipuler des blocs de coefficients 8x8 ou un ensemble de blocs de coefficients 4x4. Un contrôleur gère le transfert de données entre la mémoire FIFO, la tâche matérielle et le reste du système. Le contrôleur génère et gère des signaux qui permettent de connaitre l'état de la tâche et le traitement en cour. Cela permet de savoir si la partition peut être reconfigurée ou si le système doit attendre la fin du traitement en cours. Avant toute reconfiguration, le module qui gère le contexte sauvegarde les données utiles si cela est nécessaire. Le contexte sauvegardé sera composé de l'indice du dernier bloc traité et des données nécessaires pour reconstituer le flux binaire compressé de tailles différentes. Le contrôleur réalisera une restauration du contexte quand la région sera reconfigurée. Le wrapper statique communique avec la tâche matérielle qui se trouve sur la partition PR par le biais d'un wrapper dynamique propre à chaque tâche matérielle. Ce wrapper comprend une interface dynamique dont la complexité dépend de la complexité des transferts de données de chaque tâche matérielle. Ce wrapper permet d'obtenir une communication identique pour chaque tâche

matérielle qui sera implémentée sur la partition PR (figure 5.6).

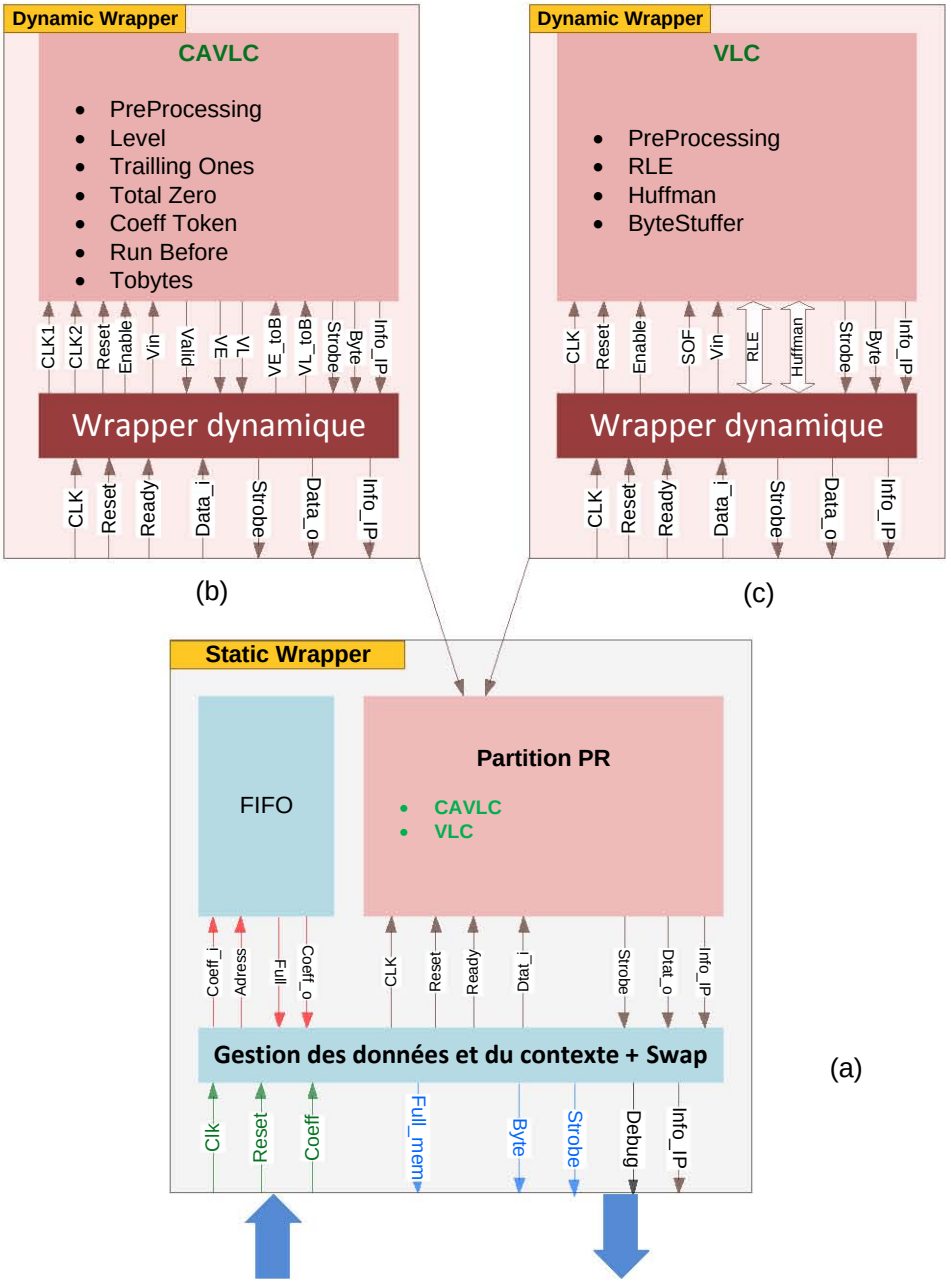


FIGURE 5.6 – Encapsulation des tâches matérielles. (a) Wrapper dynamique de CAVLC (b) Wrapper dynamique de VLC et (c) Wrapper statique de la PR partition.

5.4.3 Implémentation sans adaptation

La première architecture que nous présentons utilise le flot de conception standard (PBPR) de Xilinx, et elle ne permet pas l'adaptation de la taille des régions reconfigurables aux tâches matérielles. Cette architecture (figrure 5.7) est basée sur un bus PLB de Xilinx, les composants connectés à ce bus sont principalement utilisés pour contrôler et gérer les données d'entrées et de sorties du système et pour réaliser la reconfiguration. Cette architecture matérielle est utilisée comme un accélérateur pour les tâches CAVLC et VLC. Les bitstreams de reconfiguration sont générés en phase de conception et stockés dans une mémoire externe au système. Le processeur Microblaze permet de gérer la reconfiguration avec l'ICAP et de récupérer les bitstreams.

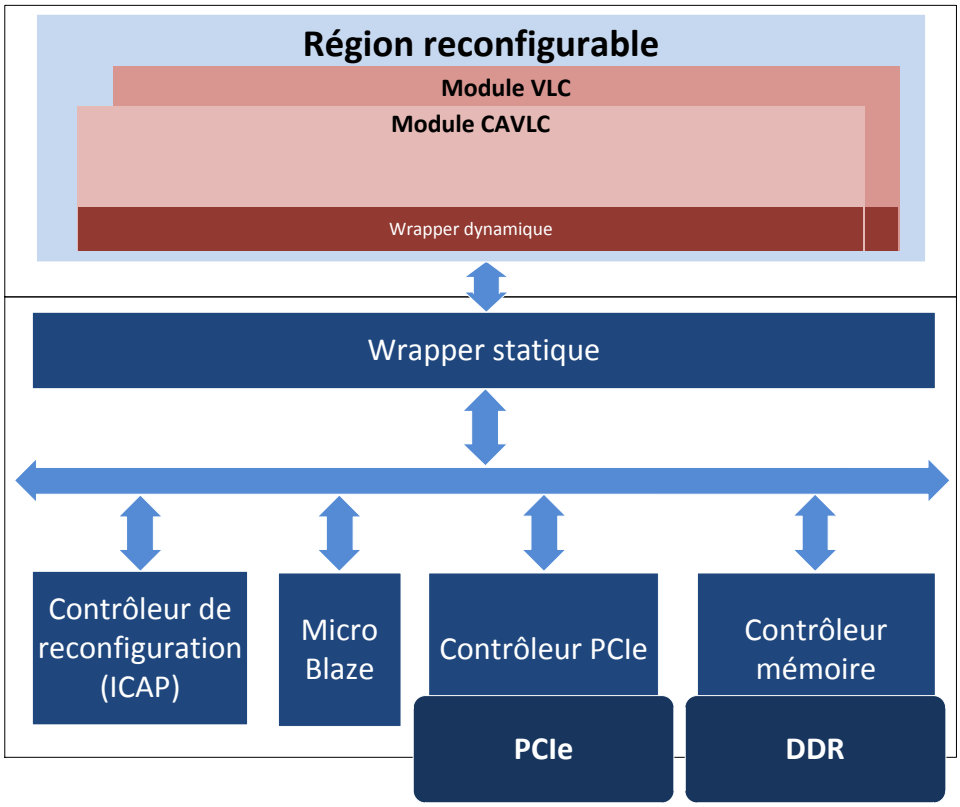


FIGURE 5.7 – Architecture sans adaptation.

La région reconfigurable a était définie en phase de conception. La forme et l'emplacement de celle-ci a été choisie en fonction des ressources requises par les tâches matérielles [MRDW11b, MRDW11a]. L'implémentation présentée (figure 5.8) utilise une région reconfigurable et cette implémentation n'est pas optimale quand l'IP CAVLC est implémentée . Pour mettre en œuvre

l'architecture proposée, nous avons ciblé un FPGA Virtex-5. Le placement de la région reconfigurable a été effectué avec l'outil PlanAhead.

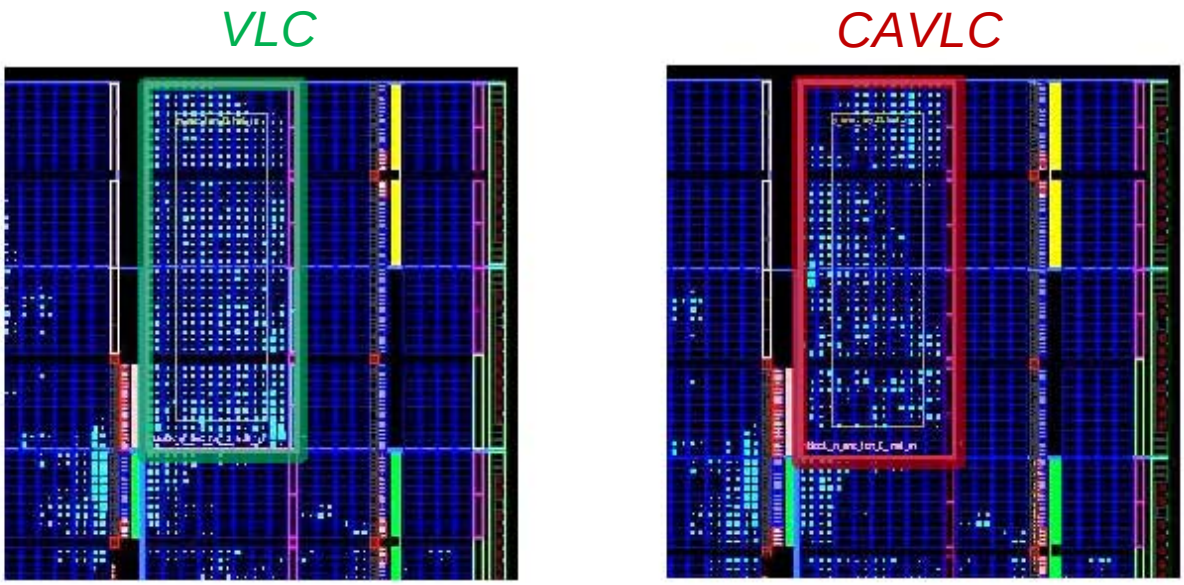


FIGURE 5.8 – Implémentation de VLC et de CAVLC sur une Architecture sans adaptation.

L'implémentation d'une architecture reconfigurable (figure 5.8) sans adapter la taille des partitions aux tâches matérielles est efficace pour VLC en terme d'utilisation des ressources. Par contre, pour CAVLC la tâche matérielle n'utilise qu'une partie des ressources de la région reconfigurable. Les ressources restantes ne sont pas utilisées et elles ne peuvent pas être utilisées par une autre tâche matérielle. Dans la partie qui suit, nous allons vous présenter une architecture qui permet d'utiliser les ressources non utilisées.

5.4.4 Implémentation avec adaptation

5.4.4.1 Architecture initiale

La seconde architecture (figrure 5.9) que nous présentons utilise notre flot de conception qui permet l'adaptation des tâches matérielles de tailles différentes. Comme l'architecture précédente, elle est basée sur un bus PLB de Xilinx, les composants connectés à ce bus sont principalement utilisés pour contrôler et gérer les données d'entrée et de sortie du système et pour réaliser la reconfiguration. La région reconfigurable a été partitionnée en deux partitions PR de

tailles identiques permettant d'implémenter les tâches matérielles pour réaliser le codage entropique. La tâche matérielle VLC utilise les deux partitions PR, et la tâche matérielle CAVLC utilise une partition PR et laisse la seconde libre.

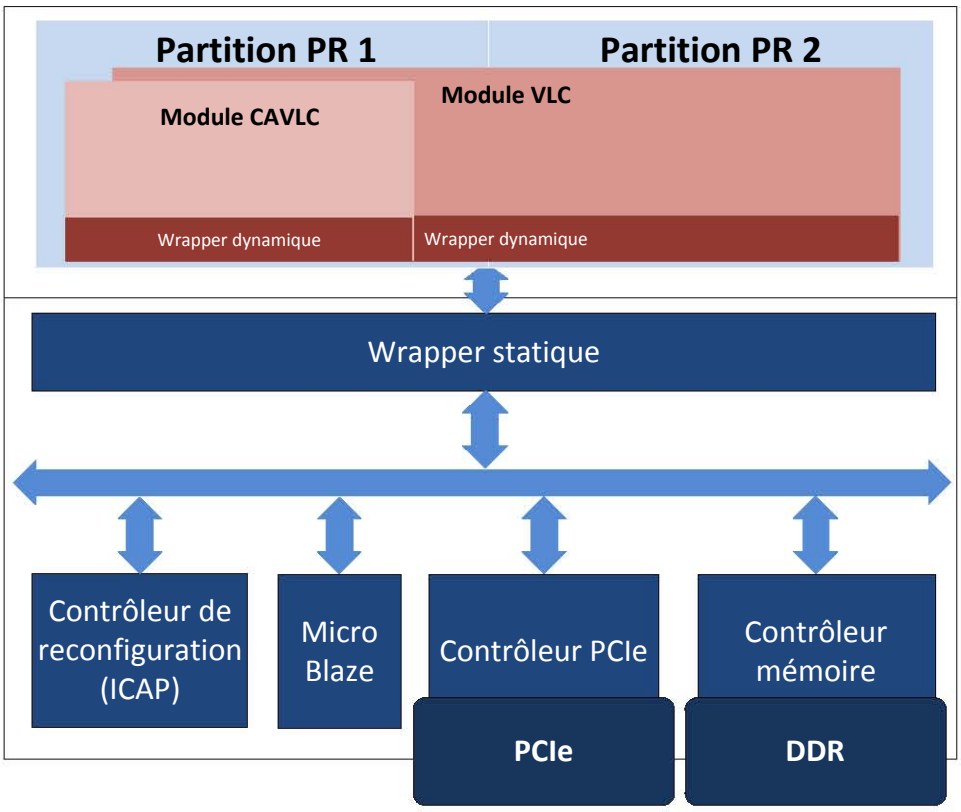


FIGURE 5.9 – Architecture avec adaptation de la région reconfigurable découpée en PR partition.

L'implémentation présentée (figrure 5.10) correspond à la première étape de notre flot de conception qui permet la mise en œuvre de la conception initiale. Dans cette conception initiale on retrouve les définitions de la région statique et de la région reconfigurable ainsi que ces deux partitions PR vides (Black Box). Chaque partition PR est composée de 8 slices de type CLB et d'une slice de type BRAM. Les interfaces de communication (pin partitions) entre la région statique et les partitions PR sont incluses dans la partie statique et elles sont automatiquement placées. Nous avons ensuite extrait les informations de placements et de routages des pin partitions pour concevoir nos sous-conceptions (étape 2 du flot de conception).

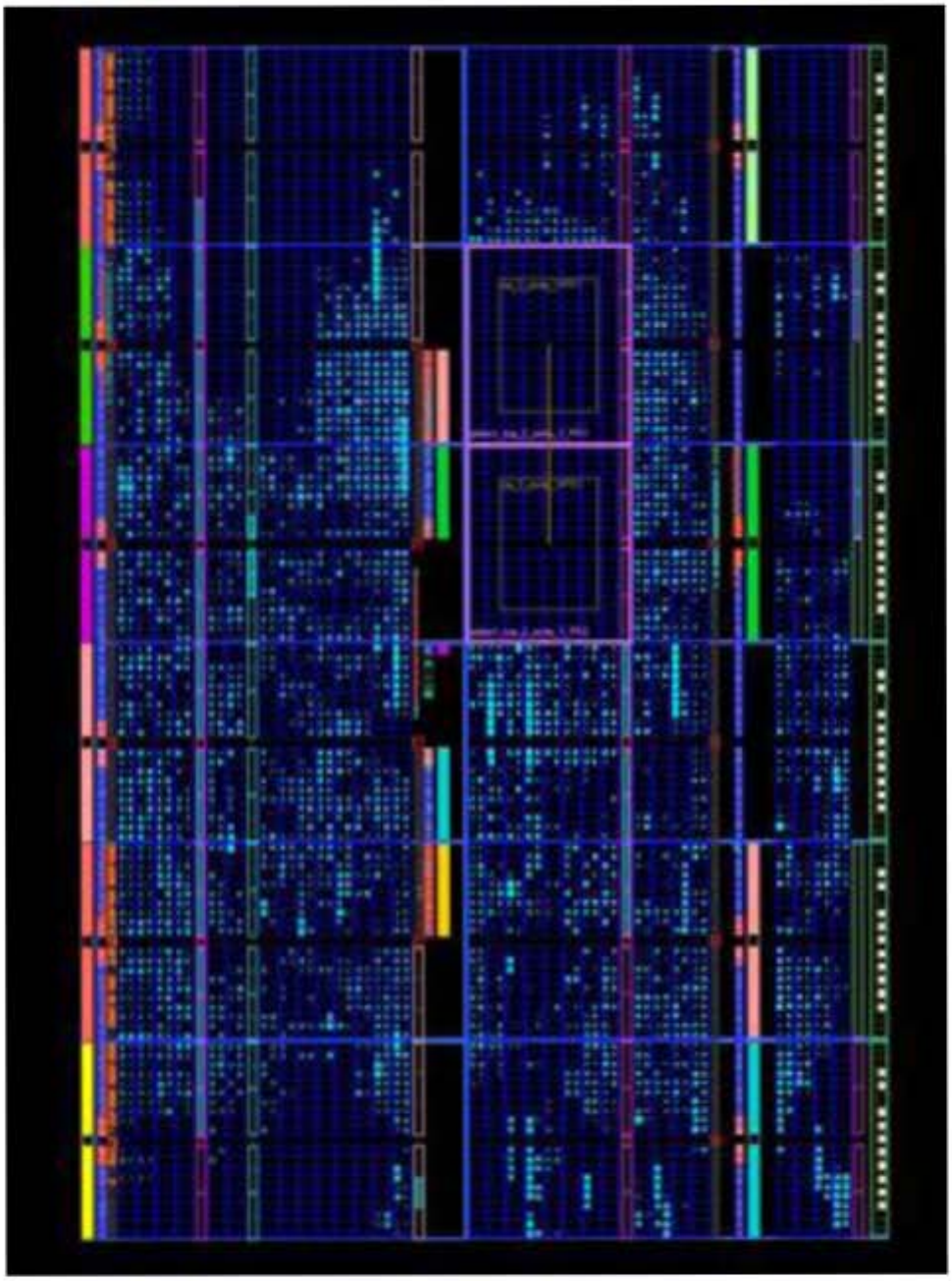


FIGURE 5.10 – Implémentation de la conception initiale de l'architecture adaptative (région statique + deux partitions PR vides « blank »).

5.4.4.2 Partitionnement de l'architecture

Nous avons ensuite créé deux sous-conceptions dans lesquels nous avons contraint le placement des pins partitions (étape 3 du flot de conception). Dans ces deux sous-conceptions, la partie statique de l'architecture n'est pas utilisée, cela permet d'alléger la charge de travail

et de réduire le nombre de fichiers à manipuler. Cependant, le fichier de contraintes du design initial est appliqué pour contraindre le placement des pins partions (fichier UCF récupéré dans l'étape 2). La figure 5.11 représente la sous-conception pour l'implémentation de la tâche matérielle VLC. Elle utilise entièrement la première partition PR par contre elle n'utilise pas la slice BRAM de la seconde partition. Ce choix permet de réduire la taille du bitstream de configuration et donc de diminuer les temps de reconfiguration. La figure 5.12 représente la sous-conception pour l'implémentation de la tâche matérielle CAVLC. Elle n'utilise qu'une seule partition PR sans les BRAM. La partition PR libre peut être utilisée pour implémenter un second codeur entropique CAVLC ou toutes autres tâches matérielles ayant un wrapper de connection compatible avec la partition PR.

Pour finaliser notre architecture globale nous devons réaliser le reroutage des pins partitions (étape 4 du flot de conception) pour les rendre communes à toutes les sous-conceptions. Il faut donc récupérer le routage des pins partitions de la conception initiale à partir du fichier XDL. Dans ce fichier, on extrait l'information de connexion des pins partitions sur les différentes LUT. Dans l'exemple en annexe 5, la pin partition est routée sur l'entrée A2 de la LUT. Il faut ensuite modifier la sous-conception qui a un routage différent (la pin partition est routée sur l'entrée A1 de la LUT). Une fois la modification effectuée, le routage est identique pour les deux sous-conceptions et pour la conception initiale.

5.4.5 Validation expérimentale

5.4.5.1 Gain obtenu en surface sur l'architecture complète

Grâce à la reconfiguration dynamique, nous pouvons obtenir un gain sur l'utilisation des ressources logiques par rapport à une implémentation statique. Le tableau 5.2 est un comparatif entre trois architectures qui utilisent les tâches matérielles CAVLC et VLC, la première est une implémentation statique des deux tâches matérielles en parallèle, la seconde correspond à une architecture avec une région reconfigurable (Implémentation sans adaptation - 5.4.3) et la dernière correspond à une architecture avec deux partitions PR (Implémentation avec adaptation - 5.4.4).

Le gain obtenu avec une architecture reconfigurable sans adaptation au niveau des ressources

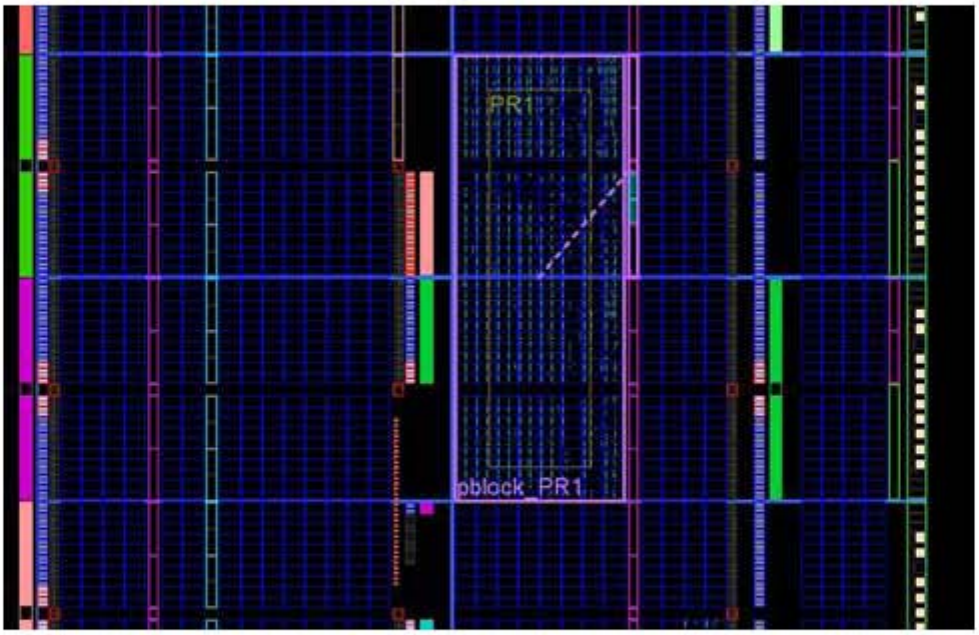


FIGURE 5.11 – Implémentation de la sous-conception avec VLC. Elle utilise deux PR partitions. La région statique n'est pas réimplémentée..

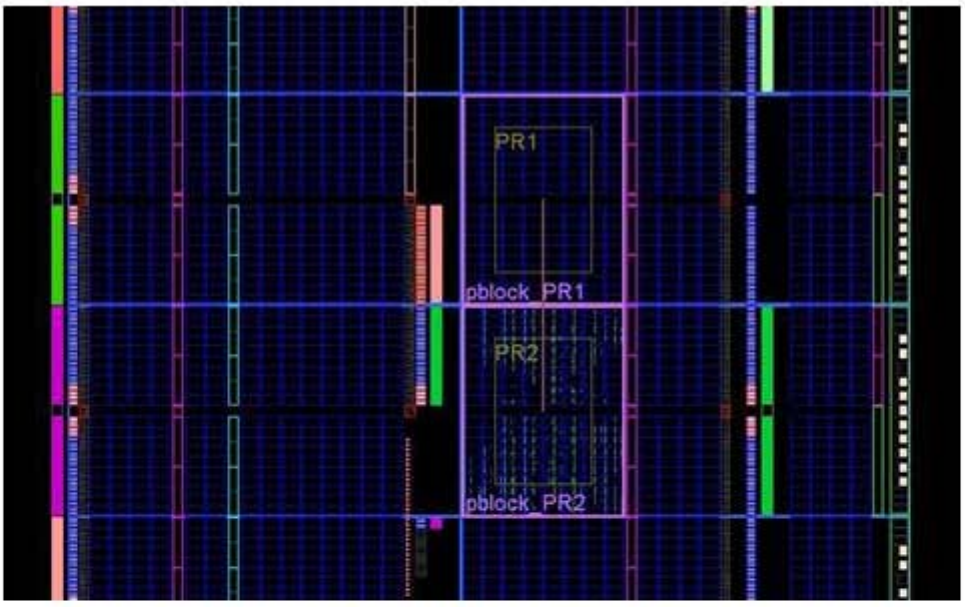


FIGURE 5.12 – Implémentation de la sous-conception avec CAVLC. Elle utilise une seule PR partitions. La région statique n'est pas réimplémentée.

logiques est d'environ 30 % par rapport à une conception statique où les deux tâches matérielles sont présentes dans le circuit. Par contre nous avons une légère augmentation au niveau des

ressources de type BRAM. Cette augmentation est obligatoire en utilisant le flot de conception PBPR de Xilinx, ce flot nous oblige à sélectionner une région rectangulaire de taille unique pour les deux tâches matérielles. CAVLC n'a pas besoin de BRAM, mais VLC en a besoin d'une seule. Il faut donc placer une région reconfigurable qui englobe des ressources de ce type. La région reconfigurable recouvre deux frames BRAM (une frame BRAM est composée de 8 BRAM 36) donc elle en utilise 8.

Grâce à notre nouveau flot de conception, nous avons pu adapter la taille de la région reconfigurable aux tâches matérielles. Pour cela nous avons créé deux partitions PR liées ou non à une slice BRAM en fonction des besoins de la tâche matérielle en mémoire. Le gain obtenu avec cette architecture adaptative dépend de la tâche matérielle implémentée sur la région reconfigurable. Plus précisément, cela dépend du nombre de partitions PR qu'occupe cette tâche. Pour VLC les gains en ressources logiques sont identiques à une architecture sans adaptions, car la fonction utilise les deux partitions PR, par contre nous avons une légère diminution au niveau des ressources de type BRAM. Ce gain est obtenu grâce à une meilleure sélection des ressources. Avec le nouveau flot de conception, nous avons utilisé une seule frame BRAM et la seconde est vide. Pour CAVLC les gains en ressources logiques sont plus importants qu'avec une architecture sans adaptions, car la fonction utilise une seule PR partition. Nous obtenons ainsi au niveau des ressources logiques une optimisation d'environ 46 % par rapport à conception statique où les deux tâches matérielles sont présentes sur le circuit. Pour finir, nous obtenons un gain de 3 % sur les ressources mémoires de type BRAM.

TABLE 5.2 – Resources utilisées pour l'architecture complète

Ressources	Statique	Reconfigurable			Optimisation (%)		
		Sans adaptation	Avec adaptation		Sans adaptation	Avec adaptation	
			VLC	CAVLC		VLC	CAVLC
LUT	7398	5598	5598	4318	25 %	25 %	41 %
Slice L	2380	1475	1475	1155	38 %	38 %	51 %
Bram36	26	33	29	25	-26 %	-11 %	3 %

Le partitionnement de la région reconfigurable nous a permis de réaliser une meilleure sélection ressources mémoire par rapport à une architecture reconfigurable sans partitionnement. De plus dans le cas où CAVLC est implémenté nous avons diminué les ressources utilisées.

Dans la partie qui suit, une étude sur l'efficacité en surface des tâches matérielles est réalisée pour déterminer le gain obtenu grâce à une architecture avec adaptation.

5.4.5.2 Étude de l'efficacité en surface

Nous avons défini précédemment une équation de l'efficacité en surface (équation 5.2). Cette équation est exprimée en fonction de la taille en frame qui correspond aux ressources reconfigurables utilisées. Les ressources reconfigurables sont de trois types (CLB, BRAM et DSP48), chaque ressource est rassemblée sur des frames (plus petit élément reconfigurable d'un FPGA). Le nombre de ressources qui constitue une frame dépend du FPGA utilisé, à titre d'exemple le FPGA choisi pour valider nos travaux est un Virtex 5 de Xilinx. Pour ce FPGA une frame CLB équivaut à 160 LUT et 40 slices ; une frame BRAM équivaut à 4 mémoire BRAM36 et une frame DSP équivaut à 8 DSP48E. Nous pouvons donc définir trois efficacités en surface secondaires, une pour chaque type de frame Es (CLB) (équation 5.3), Es (BRAM) (équation 5.4) et Es (DSP) (équation 5.5). Es correspond à la moyenne des sous-efficacités présentes sur la région reconfigurable ou sur la partition PR , soit un ensemble de frames CLB, BRAM et DSP (équation 5.6)) ou un ensemble CLB et BRAM (équation 5.7) ou un ensemble CLB et DSP (équation 5.8) ou des CLB seules (équation 5.9). Tous les autres cas sont impossibles, car une région reconfigurable ou une partition PR doit au moins contenir une frame CLB.

$$Es_{CLB} = \frac{TTM_{CLB}}{TRR_{CLB}} \tag{5.3}$$

$$Es_{BRAM} = \frac{TTM_{BRAM}}{TRR_{BRAM}} \tag{5.4}$$

$$Es_{DSP} = \frac{TTM_{DSP}}{TRR_{DSP}} \tag{5.5}$$

$$Es1 = \frac{N_{FrameCLB} * ES_{CLB} + N_{FrameBRAM} * ES_{BRAM} + N_{FrameDPS} * ES_{DSP}}{N_{Frametotal}} \tag{5.6}$$

$$Es2 = \frac{N_{FrameCLB} * ES_{CLB} + N_{FrameBRAM} * ES_{BRAM}}{N_{Frametotal}} \tag{5.7}$$

$$Es3 = \frac{N_{FrameCLB} * ES_{CLB} + N_{FrameDPS} * ES_{DSP}}{N_{Frametotal}} \tag{5.8}$$

$$Es4 = ES_{CLB} \tag{5.9}$$

Le tableau 5.3 montre l'efficacité des tâches matérielles CAVLC et VLC. Avec une architecture reconfigurable basique, l'efficacité de VLC est de 83.16 % et pour CAVLC elle est de 40.8 %. L'efficacité est faible pour ce type d'architecture, ceci est dû à l'obligation de sélectionner suffisamment de ressources pour que la région reconfigurable soit compatible avec les deux tâches matérielles. Pour une architecture adaptative, l'efficacité de VLC augmente à 88 %, ce faible gain de 4.84 % est dû à une meilleure sélection des ressources de type BRAM (les autres ressources ne peuvent pas être améliorées). Par contre pour CAVLC l'utilisation du partitionnement nous permet d'obtenir une efficacité de 93 %. De plus nous avons une partition PR qui est entièrement libre et nous avons libéré toutes les ressources de type BRAM.

TABLE 5.3 – Resources utilisées par les tâches matérielles sur la région reconfigurable

Efficacité	Utilisation CAVLC (%)		Utilisation VLC (%)	
	Sans adaptation	Avec adaptation	Sans adaptation	Avec adaptation
ES CLB	46 %	93 % + 1PR libre	92 %	92 %
ES BRAM	0 %	-	12,5 %	25 %
ES	40,8 %	93 % + 1PR libre	83.16 %	88 %

5.4.5.3 Gain obtenu sur le temps de reconfiguration

Le FPGA Virtex 5 de Xilinx se reconfigure par frame. Une frame CLB (160 LUT et 40 slices) est équivalente à un bitstream de 5904 octets. Sachant que la reconfiguration peut se faire à une vitesse allant jusqu'à 50 Mo/s, une frame CLB prend 118 μs à se reconfigurer. Pour une frame BRAM, sa taille est de 24 912 octets, elle nécessite donc 863 μs pour se reconfigurer. Donc le temps de reconfiguration dépend de la taille des régions reconfigurables ou des partitions PR . Par conséquent, dans une architecture reconfigurable basique (flot PBPR de Xilinx), le temps de reconfiguration sera identique pour toute tâche matérielle qui utilise la même région reconfigurable. Dans notre implémentation, le temps de reconfiguration pour VLC et CAVLC

est de 3,61 ms (tableau 5.4). Avec notre flot de conception, nous avons pu diviser la région reconfigurable en partitions PR, cette adaptation de l'architecture nous permet de sélectionner le nombre de partitions PR utilisées en fonction des tâches matérielles implémentées. Cette sélection nous permet dons d'optimiser la taille des bitstreams en fonction des tâches matérielles et du nombre de partitions PR qu'elles occupent. Dans notre implémentation adaptative, le temps de reconfiguration pour VLC est passé à 2,75 ms. Pour CAVLC il est de 0,94 ms auquel on peut ajouter le temps de reconfiguration de la seconde partition PR pour la rendre vide, ce qui nous fait un temps total de 1,88 ms. L'adaptation nous a donc permis de diminuer les temps de reconfiguration.

TABLE 5.4 – Évaluation du temps de reconfiguration

	Sans adaptation	Avec adaptation	
		VLC	CAVLC
Frame	18 (16 CLB ; 2 Bram)	17 (16 CLB ; 1 Bram)	8 par PR (8 CLB)
Temps de reconfiguration	3.61ms (16x118μ + 2x863μ)	2.75 ms (16x118μ + 863μ)	1.88 ms (8x118μx2PR)

5.5 Étude de l'impact du partitionnement sur le gain en surface

5.5.1 Contexte de l'étude

Dans la partie précédente, la méthode et l'architecture proposées ont été validées sur une application réelle avec deux tâches. Dans cette partie, nous présentons une étude sur le gain en surface en fonction des tâches implémentées. Comme le montre la figure 5.13, le gain en surface dépend de la tâche implémentée et de la finesse du partitionnement. Quand la tâche la plus grande est implémentée, le gain est nul (figure 5.13 T1). Par contre quand une tâche plus petite est implémentée le gain en surface peut être utilisé pour réaliser du parallélisme de tâches, sur la figure 5.13 quand T2 est implémentée sur l'architecture partitionnée l'espace libre est utilisé pour implémenter T5 et T4 en parallèle. Le gain en surface peut être aussi utilisé pour réali-

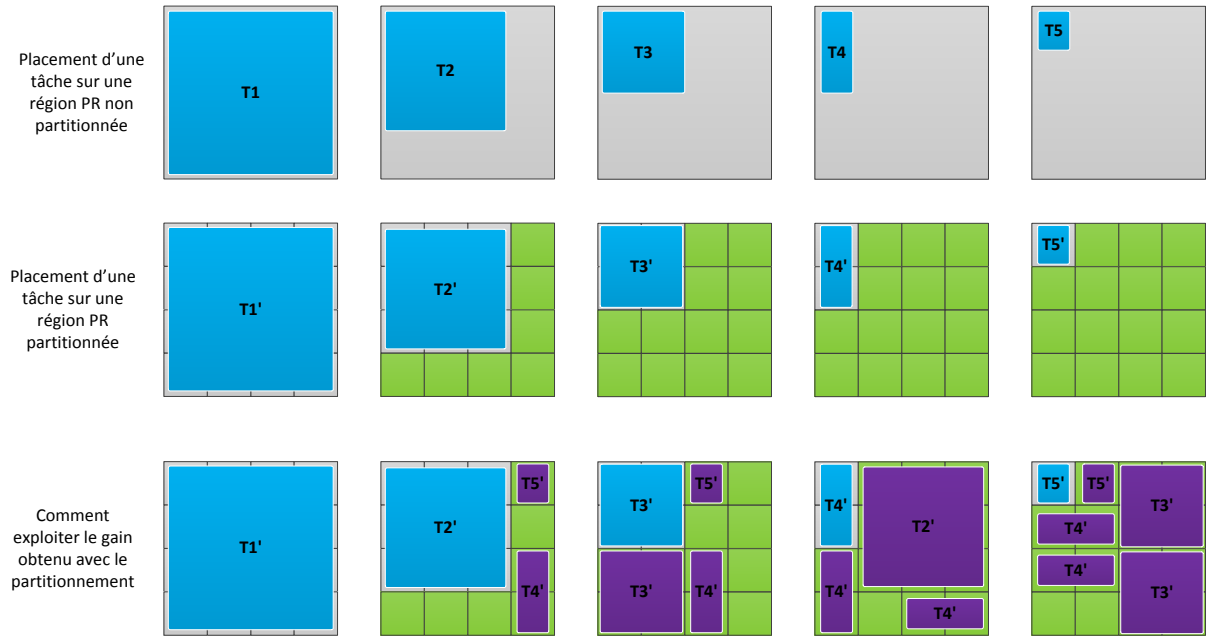


FIGURE 5.13 – Gain en surface obtenues avec le partitionnement.

ser de la redondance de tâches sur la figure 5.13 quand T5 est implémentée sur l'architecture partitionnée, l'espace libre est utilisé pour implémenter T3 et T4 en parallèle. Pour évaluer l'impact de la méthode de partitionnement qui permet ce gain en surface, il est nécessaire d'étudier plusieurs applications avec plusieurs tâches matérielles de tailles différentes. L'étude doit aussi prendre en compte la méthode que nous avons proposée et le surcoût en ressources qu'elle implique. Pour chaque application, tous les partitionnements possibles de la région reconfigurable devront être étudiés.

5.5.2 Prise en compte du surcoût de la méthode proposée

La méthode que nous avons proposée implique un surcoût sur les interfaces de connexion. Elle implique de réserver des ressources d'interconnexion pour chaque partition. Ces ressources permettent d'assurer le placement des pins partitions et la connexion de la partition PR au bus pour qu'elle communique avec l'extérieur. Les ressources utilisées pour cette communication ne peuvent pas être utilisées pour autre chose. Le surcoût est d'une LUT par pin partition. Nous avons évalué ce surcoût pour une architecture implémentée sur un Virtex 5. Sur ce type de FPGA, les éléments de calcul communiquent par l'intermédiaire d'un BUS PLB de 32bits de

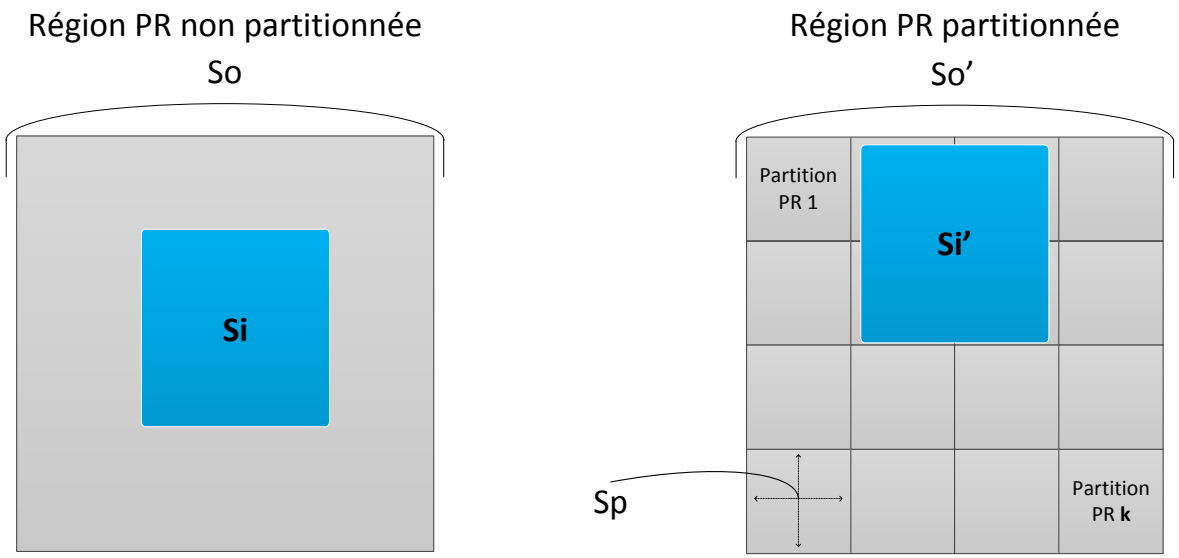


FIGURE 5.14 – Illustration des termes utilisés pour définir le critère d'évaluation.

données, de 32 bits d'adresse et de 26 bits de contrôle ce qui fait un total de 90 bits. Chaque bit est un point de connexions (une Pin Partition). Les Pins Partition utilisent une LUT par point de connexion et une frame contient 160 LUTs. Par conséquent, le surcoût est équivalent à 60 % d'une frame par partition pour cet exemple. Dans la suite de l'étude, nous fixerons le surcoût à 0.6 et nous le nommerons w.

5.5.3 Critère d'évaluation

Le critère d'évaluation de cette étude est le gain en surface obtenu grâce au partitionnement. Ce critère nous permet d'évaluer l'impact du partitionnement d'une région reconfigurable sur l'efficacité en surface et de comparer le gain en surface entre une architecture partitionnée et une architecture non partitionnée (figure 5.14).

Dans le cas d'une région non partitionnée, une tâche matérielle (i) occupe entièrement la surface de la région reconfigurable (So), même si la surface de la tâche (Si) est plus petite que la surface de la région reconfigurable. De plus (So) doit être capable d'accueillir la plus grande des tâches i (équation 5.10). Pour une région reconfigurable partitionnée, la surface (So') de la région peut être supérieure à celle d'une région non partitionnée, cette augmentation est due au surcoût des interfaces de communication (w). Il en est de même pour les taches (i').

La région PR est partitionnée en (k) partitions PR, et la surface d'une partition PR (Sp) correspond au rapport entre sur la surface de la région et le nombre de partitions (équation 5.11).

La surface partitionnée occupée par une tâche (Si') occupe un ensemble de partitions PR, ce nombre de partitions correspond à (n), il faut prendre en compte les wrappers non utilisés sur les partitions (équation 5.12). On peut aussi définir (n) (équation 5.13) comme le rapport entre la surface partitionnée occupée par une tâche (So') et la surface d'une partition PR (Sp), ce rapport est exprimé en frames, une frame n'étant pas divisible (n) sera obligatoirement un entier arrondi à la valeur supérieure.

Pour finir, le gain obtenu pour une tâche (Δi) est la différence entre la surface de la région reconfigurable (So) et la surface partitionnée occupée par une tâche (Si'), en fonction des équations précédentes nous pouvons déterminer (Δi) (équation 5.14). Le gain (Δi(%)) (équation 5.15) est notre critère d'évaluation du partitionnement, ce critère varie en fonction de (k) qui correspond au nombre de partitions.

- Δi = 0% pas de gain en surface pour la tâche i ;
- Δi > 0% gain en surface avec le partitionnement pour la tâche i ;
- Δi < 0% perte en surface avec le partitionnement pour la tâche i.

$$So = \textbf{maxsurface}(Si) \tag{5.10}$$

$$Sp = \frac{So'}{k} \tag{5.11}$$

$$Si' = n * Sp - w * (n - 1) \tag{5.12}$$

$$\begin{aligned} n &= \textbf{arrondisup}(\frac{Si'}{Sp}) \\ &= k * \textbf{arrondisup}(\frac{Si'}{So'}) \end{aligned} \tag{5.13}$$

$$
\begin{aligned}
\Delta i &= So - Si' \\
&= So - [n * Sp - w * (n-1)] \\
&= So - [(n * \frac{So'}{k} - w * (n-1)] \\
&= So - [(\textbf{arrondisup}(\frac{Si'}{So'}) * So' - w(k * \textbf{arrondisup}(\frac{Si'}{So'}) - 1)] \\
&= So - [\textbf{arrondisup}(\frac{Si'}{So'})(So' - w * k) + w] \qquad (5.14)
\end{aligned}
$$

$$
\Delta i(\%) = \frac{\Delta i}{So} * 100 \qquad (5.15)
$$

5.5.4 Cas d'étude

Le but étant de valider une méthode permettant d'obtenir un gain en surface en fonction des tâches matérielles de tailles différentes, nous avons choisi dix applications contenant chacune dix tâches matérielles (TM i). Pour cette étude, les tâches matérielles sont définies par leur taille en frame (la frame correspond au plus petit élément reconfigurable d'un FPGA). Nous n'avons pas utilisé une application définie avec des tâches matérielles existantes, mais nous avons préféré choisir des applications générales qui contiennent chacunes un ensemble de tâches matérielles dont les tailles sont choisies aléatoirement avec une fonction aléatoire. À part pour les applications 1, 2, 3 (figure 5.15) qui sont choisies pour tester l'impact du partitionnement sur des cas particuliers.

5.5.5 Résultat de l'étude par application

Pour chaque application, nous avons étudié les partitionnements possibles de la région reconfigurable(figure 5.16, 5.17, 5.18). On peut remarquer qu'un trop grand nombre de partitions n'est pas intéressant, surtout si les partitions ont une taille proche de la taille de l'interface de communication. Dans les cas d'application présentés, l'interface de communication utilise 60 % d'une frame, si la partition a aussi la taille d'une frame, l'espace libre est de 40 % par frame. Ce type de partitionnement n'est donc pas idéal, car les interfaces de communications occupent plus de place que les tâches matérielles. Pour toutes les applications, nous avons une perte avec ce type de partitionnement. Par contre les autres types de partitionnement nous permettent

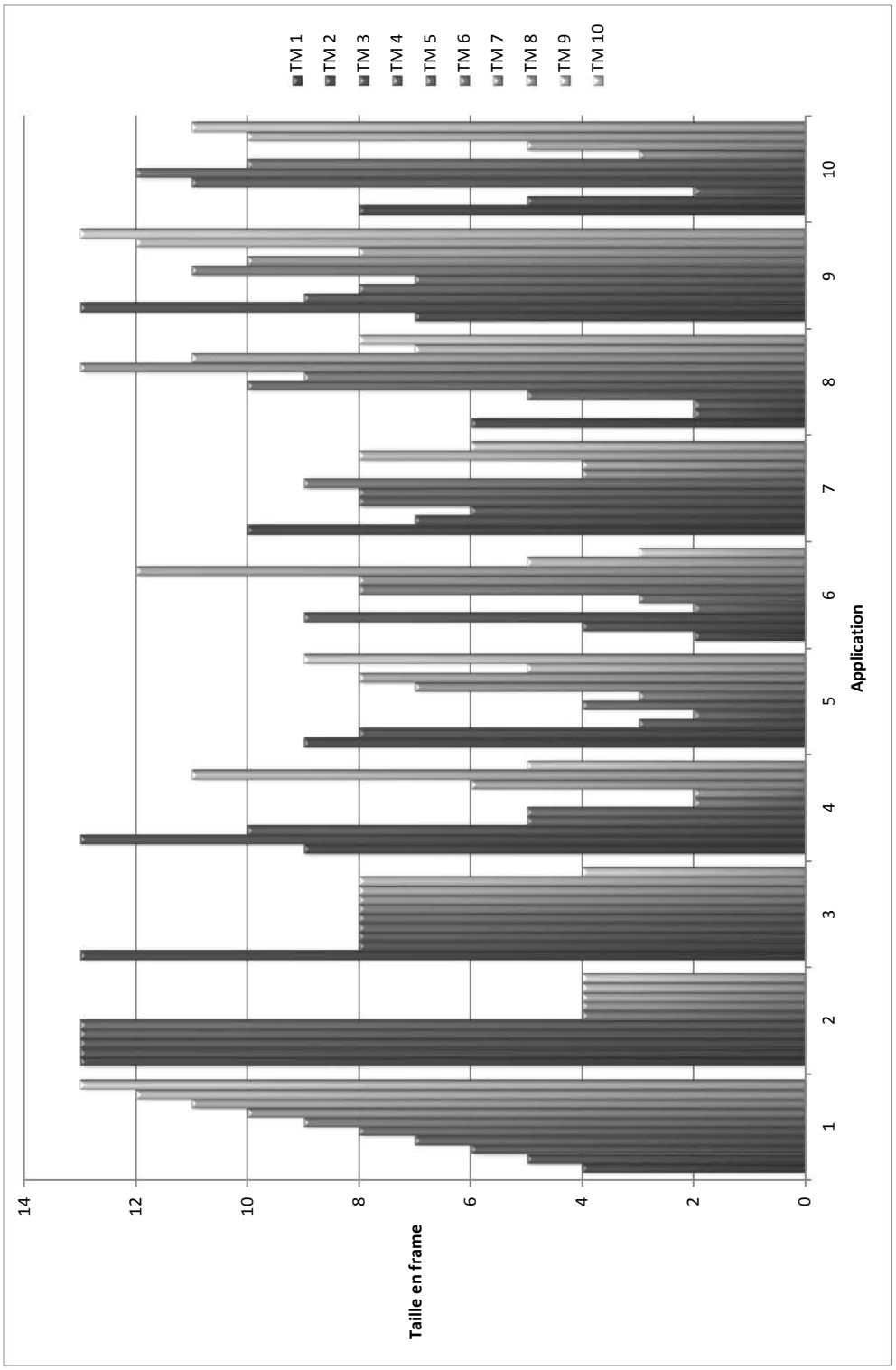


FIGURE 5.15 – Taille en frame des tâches matérielle pour chaque application.

d'avoir un gain plus ou moins important en fonction de la surface occupée par une tâche. Ce gain peut être utilisé pour augmenter le parallélisme des tâches ou pour augmenter la fiabilité d'un système en ajoutant de la redondance (figure 5.13). Pour conclure sur cette étude, on remarque que notre méthode permet un gain en surface par rapport à une architecture reconfigurable non partitionnée. Ce gain dépend de l'application, des tâches matérielles et du nombre de partitions. Pour choisir le partitionnement qui correspond à une application, il faut aussi prendre en compte l'ordonnancement des tâches. Il est donc nécessaire de réaliser une étude comparative, qui permet de déterminer les gains obtenus pour chaque tâche en fonction du partitionnement choisi. Ainsi on pourra choisir le partitionnement qui correspond le mieux à une application.

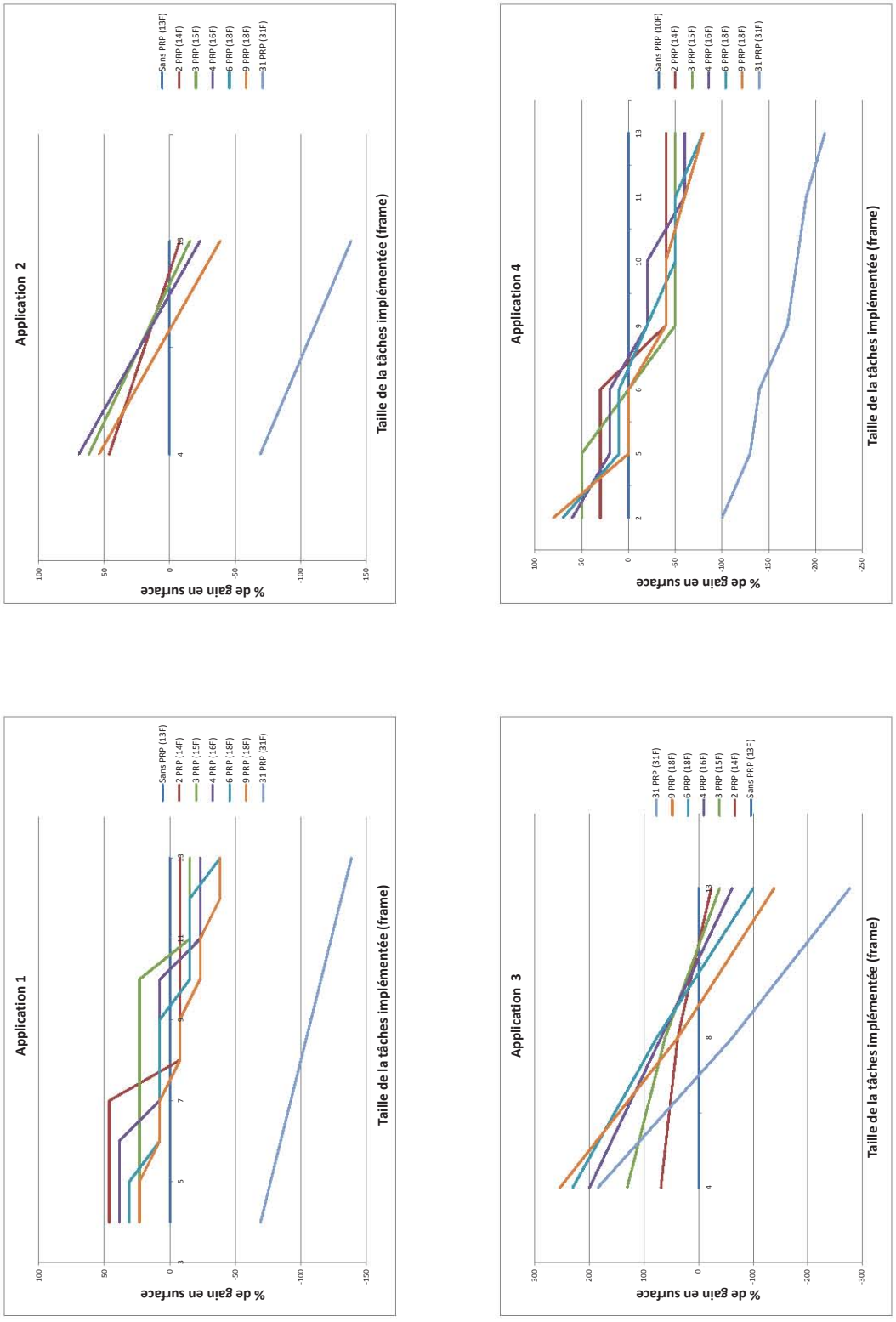


FIGURE 5.16 – Résultat sur le gain en surface pour les applications 1, 2, 3 et 4 (PRP : Partition Reconfigurable Partiellement / F : frames).

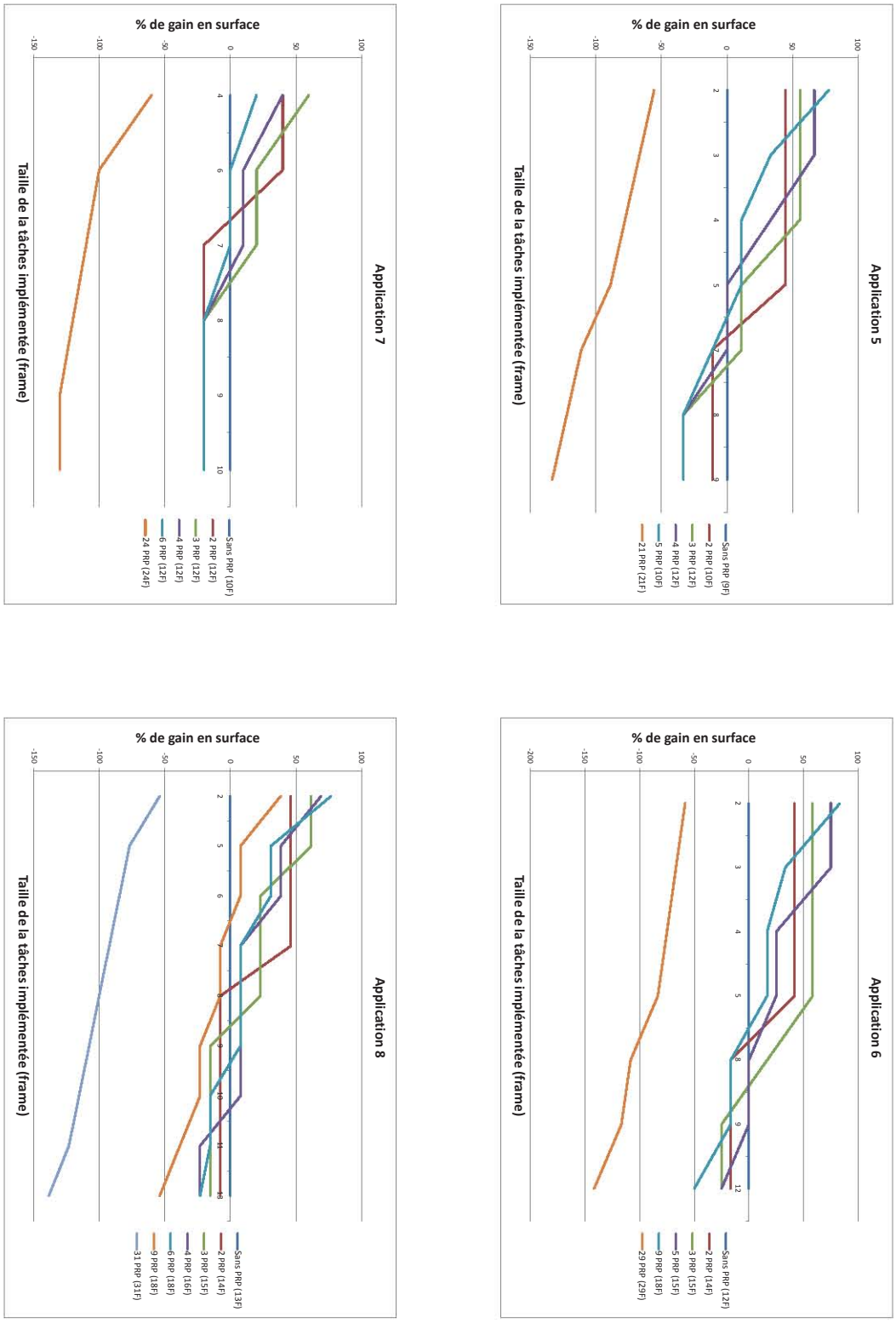
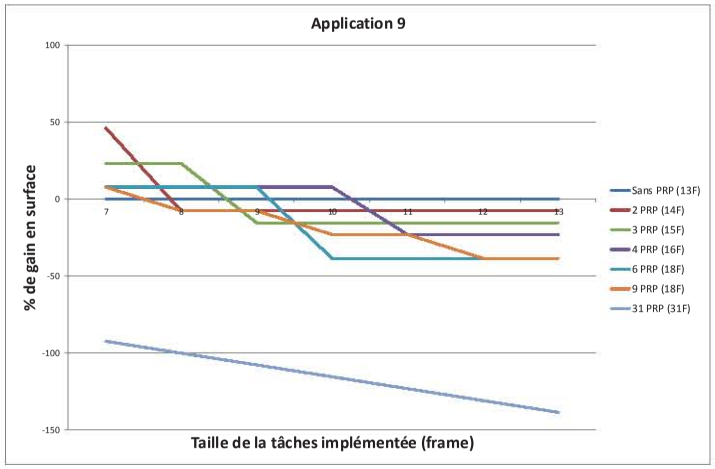


FIGURE 5.17 – Résultat sur le gain en surface pour les applications 5, 6, 7 et 8 (PRP : Partition Reconfigurable Partiellement / F : frames).



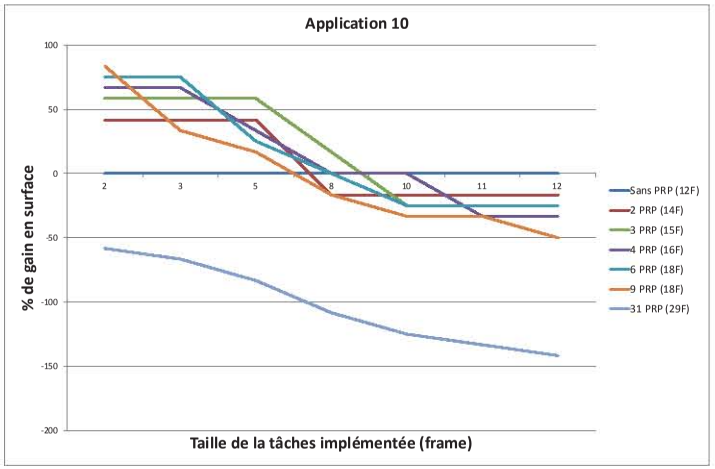


FIGURE 5.18 – Résultat sur le gain en surface pour chaque pour les applications 9 et 10 (PRP : Partition Reconfigurable Partiellement / F : frames).

5.6 Conclusion

Nous avons présenté dans ce chapitre la conception et l'implantation d'une architecture adaptative appliquée au traitement vidéo et basée sur les concepts de partitionnent d'un FPGA reconfigurable. Nous avons explicité à travers cet exemple d'application, la stratégie de mise en œuvre de notre méthode de partitionnement et nous avons validé expérimentalement la solution d'implantation matérielle du partitionnement. En effet, nous avons montré que le partitionnent apportait un gain sur l'efficacité en surface en fonction des tâches matérielles implémentées.

Pour évaluer le gain en surface que nous apporte le partitionnement d'une architecture, nous avons présenté une étude en fonction de plusieurs applications. Cette étude prend en compte la méthode de partitionnement et le flot de conception que nous avons proposé et le surcoût qu'elle implique. Pour chaque application nous avons étudié les partitionnements possibles de la région reconfigurable. Cette étude nous permet d'évaluer l'impact du partitionnement sur le gain en surface. Dans les dix cas d'étude présentés, nous avons remarqué que le gain dépend de la surface occupée par une tâche. Par conséquent, ce gain peut être utilisé pour augmenter le parallélisme des tâches ou pour augmenter la fiabilité d'un système en ajoutant de la redondance de tâches. Ces résultats démontrent la viabilité de la solution architecturale proposée.

Comme nous l'avons expliqué dans l'introduction de cette thèse, la majorité des travaux existants sur le placement portent sur la recherche et l'amélioration des performances des algorithmes de placement. Cependant, la problématique sous-jacente du partitionnement physique des FPGA et de la génération efficace des régions reconfigurables pour accueillir des tâches matérielles de différentes tailles, est très peux explorée. Avec notre méthode, nous proposons une solution pour répondre à cette problématique. Par ailleurs, associée à des techniques de placement et d'ordonnancement, notre méthode permettra d'obtenir des solutions efficaces et réalistes.

Conclusion générale et perspective

Les architectures reconfigurables à base de FPGA sont capables de modifier le comportement d'une partie du système pendant que le reste du circuit continue de s'exécuter normalement. Ces architectures, bien qu'elles évoluent d'une façon continue, souffrent encore d'un manque d'efficacité sur le placement des tâches matérielles de tailles différentes. De plus, la majorité des travaux existants portent sur la recherche d'algorithmes efficaces pour le placement et l'ordonnancement en considérant les FPGA comme une matrice uniforme et ne prennent pas en compte les contraintes réelles pour le placement des tâches matérielles et la communication entre elles.

Dans cette thèse nous avons proposé une nouvelle méthodologie de partitionnement basée sur l'utilisation d'une couche intermédiaire entre le FPGA et l'application qui permet le placement efficace des tâches matérielles de tailles différentes sur des partitions reconfigurables fixes. À partir de l'analyse des caractéristiques et des techniques de la reconfiguration dynamique des FPGA, nous avons établi les principes de l'adaptation des partitions à la taille des tâches matérielles. Ainsi, nous avons pu déterminer qu'actuellement le partitionnement des régions reconfigurables n'était pas possible avec les flots de conception standards. Nous avons aussi déterminé que les interfaces de communication entre les partitions jouent un rôle primordial dans la conception de tels systèmes. En effet, un moyen de communication robuste et adapté à la reconfiguration d'un système doit être utilisé. La méthode et la solution architecturale proposées permettent de répondre aux contraintes de placement des tâches matérielles de tailles différentes sur des partitions reconfigurables. Cette méthode consiste à définir des parti-

tions de tailles fixes capables d'accueillir des tâches matérielles et d'assurer la communication avec le reste du système. L'ensemble des points présentés dans ces travaux de thèse a été validé au niveau expérimental sur un exemple concret d'application de traitement d'images vidéo. En effet, nous avons présenté un exemple pratique mettant en oeuvre la conception et l'implantation d'une architecture avec des tâches matérielles de tailles différentes. Nous avons réalisé une validation expérimentale du système global par son implantation réelle sur une plateforme. Nous avons montré sur cette application que le partitionnement de la région reconfigurable est possible et qu'il apporte un gain significatif sur l'efficacité en surface en fonction des tâches matérielles implémentées par rapport à une architecture reconfigurable non partitionnée. Ce gain est obtenu grâce à la possibilité de placer une tâche matérielle sur une ou plusieurs partitions reconfigurables. Pour évaluer le partitionnement d'une architecture, nous avons présenté une étude sur le gain en surface en fonction de plusieurs applications. Cette étude nous a permis d'évaluer l'impact du partitionnement sur le gain en surface. Les résultats obtenus ont mis en évidence la dépendance entre la taille optimal du partitionnement pour le FPGA et la surface occupées des tâches matérielles de l'application.

Une poursuite possible de ces travaux est de prendre en compte l'ordonnancement des tâches pour choisir le partitionnement des régions reconfigurables. L'idée serait de prendre en compte le paramètre "temps d'exécution" (temps de présence de la tâche dans la région reconfigurable) pour optimiser encore davantage les ressources. Le critère "temps d'exécution" pourrait être associé à celui de "surface libérée" afin de maximiser le couple des ressources disponibles et la durée de disponibilité.

De plus il serait intéressant d'explorer des solution de communication basées sur les réseaux sur puce (NoC). Des travaux sur un réseau programmable sur puce (QNoC) pourrait avantageusement faciliter l'interopérabilité des tâches matérielles reconfigurables dynamiquement. En termes d'application, il serait intéressant d'évaluer l'apport de notre méthode sur des systèmes de tolérance aux fautes, en effet on pourrait optimiser le nouveau placement d'une tâche matérielle dans le cas où un défaut apparaitrait.

Annexes

Sommaire

1 Définition d'une hard macro

Une Hard Macro est un bloc complètement prérouté et préplacé qui contient des éléments logiques et des interconnexions. Une Hard Macro peut être instanciée à plusieurs reprises et peut être verrouillée à des endroits différents à l'intérieur du FPGA. Elle permet de reproduire des parties d'une conception exactement de la même façon tout au long des différentes versions de la conception. Comme les éléments statiques des différents scénarios sont fonctionnellement identiques et que tous les éléments statiques sont ressemblés par la Hard Macro, cette approche permet une conception cohérente de l'architecture.

2 Description VHDL d'une région reconfigurable avec 6 PR partitions

```
----------------------------------------------------------------------------
– Description : Sous conception avec 6 PR partitions (61 à 66)
– Module Name : SSD_2_PR_Part
----------------------------------------------------------------------------
library IEEE ;
use IEEE.STD_LOGIC_1164.ALL ;
use UNISIM.VComponents.all ;
entity SSD_6_PR_part is port(
data_i_61 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_61 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_62 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_62 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_63 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_63 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_64 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_64 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_65 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_65 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_66 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_66 : out STD_LOGIC_VECTOR (0 to N)) ;
end SSD_6_PR_part ;
architecture Behavioral of SSD_6_PR_part is
COMPONENT PR_part_6 PORT(
data_i_6 : in STD_LOGIC_VECTOR (0 to N) ;
```

```
data_o_6 : out STD_LOGIC_VECTOR (0 to N)) ;
END COMPONENT ;
begin
Inst_PR_part_61 : PR_part_6 PORT MAP(
data_i_6 =>data_i_61,
data_o_6 =>data_o_61) ;
Inst_PR_part_62 : PR_part_6 PORT MAP(
data_i_6 =>data_i_62,
data_o_6 =>data_o_62) ;
Inst_PR_part_63 : PR_part_6 PORT MAP(
data_i_6 =>data_i_63,
data_o_6 =>data_o_63) ;
Inst_PR_part_64 : PR_part_6 PORT MAP(
data_i_6 =>data_i_64,
data_o_6 =>data_o_64) ;
Inst_PR_part_65 : PR_part_6 PORT MAP(
data_i_6 =>data_i_65,
data_o_6 =>data_o_65) ;
Inst_PR_part_66 : PR_part_6 PORT MAP(
data_i_6 =>data_i_66,
data_o_6 =>data_o_66) ;
end Behavioral ;
```

3 Description VHDL d'une région reconfigurable avec 1 PR partition

```
----------------------------------------------------------------------------------
– Description : Sous conception avec 1 PR partitions (11)
– Module Name : SSD_1_PR_Part
----------------------------------------------------------------------------------
library IEEE ;
use IEEE.STD_LOGIC_1164.ALL ;
use UNISIM.VComponents.all ;
entity SSD_1_PR_part is port (
data_i_11 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_11 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_12 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_12 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_13 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_13 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_14 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_14 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_15 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_15 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_16 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_16 : out STD_LOGIC_VECTOR (0 to N)) ;
end SSD_1_PR_part ;
architecture Behavioral of SSD_1_PR_part is
COMPONENT PR_part_1 PORT(
data_i_11 : in STD_LOGIC_VECTOR (0 to N) ;
```

```
data_o_11 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_12 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_12 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_13 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_13 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_14 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_14 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_15 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_15 : out STD_LOGIC_VECTOR (0 to N) ;
data_i_16 : in STD_LOGIC_VECTOR (0 to N) ;
data_o_16 : out STD_LOGIC_VECTOR (0 to N)) ;
END COMPONENT ;
begin
Inst_PR_part_11 : PR_part_1 PORT MAP(
data_i_11 =>data_i_11,
data_o_11 =>data_o_11,
data_i_12 =>data_i_12,
data_o_12 =>data_o_12,
data_i_13 =>data_i_13,
data_o_13 =>data_o_13,
data_i_14 =>data_i_14,
data_o_14 =>data_o_14,
data_i_15 =>data_i_15,
data_o_15 =>data_o_15,
data_i_16 =>data_i_16,
data_o_16 =>data_o_16) ;
end Behavioral ;
```

4 Pin partition

Les informations sur l'emplacement de chaque pin partition (voir extrait UCF) regroupent son type d'élément logique (BEL) (voir extrait UCF) et les coordonnées de la SLICE (voir extrait UCF) qu'elle utilise. La commande native PR2UCF génère un fichier de contraintes utilisateurs de type UCF (users constraints files) qui regroupe ces informations. Si on veut obtenir les informations sur les pins partitions de la conception A, il faut exécuter la commande " Pr2ucf Design_A_Config_Blank.ncd –o pin_part_Design_A.ucf ".

Extrait d'un fichier UCF :

PIN "Inst_PR_part_61.data_i_61[0]" LOC = *SLICE_X38Y86 ;* |*BEL = A6LUT ;*

PIN "Inst_PR_part_61.data_i_61[1]" LOC = SLICE_X35Y90 ; |BEL = A6LUT ;

PIN "Inst_PR_part_61.data_o_61[0]" LOC = SLICE_X36Y102 ; |BEL = A6LUT ;

PIN "Inst_PR_part_61.data_o_61[1]" LOC = SLICE_X38Y100 ; |BEL = B6LUT ;

5 Reroutage d'une Pin Partition

Le reroutage des pins partitions (étape 4 du flot de conception) permet de rendre communes toutes les connexions sous-conceptions. Il faut récupérer le routage des pins partition de la conception initiale à partir du fichier XDL. Dans ce fichier, on extrait l'information de connexion de la pin partition sur la LUT. Dans l'exemple ci-dessous (voir extrait XDL CI), la pin partition est routée sur l'entrée A2 de la LUT. Il faut ensuite modifier la sous-conception qui a un routage diffèrent (voir extrait XDL SC), la pin partition est routée sur l'entrée A1 de la LUT. Une fois la modification effectuée, le routage est identique aux deux conceptions (voir extrait XDL SC rerouté).

Extrait XDL de la conception initiale :

Inst "Comp_51188" "SLICEL",placed CLBLL_X23Y86 SLICE_X39Y86...
D6LUT :top_2_zone_1/top_2_zone_1/PR1/data_i(0)_PROXY :#LUT :O6=A2 ...

Extrait XDL d'une des sous conception :

inst "PR1/Inst_cavlcto/cavlc/state_FSM_FFd2" "SLICEL",placed CLBLL_X23Y86
SLICE_X39Y86 ,... D6LUT :PR1/data_i(0)_PROXY :#LUT :O6=A1 ...

Extrait XDL SC rerouté d'une des sous conception :

inst "PR1/Inst_cavlcto/cavlc/state_FSM_FFd2" "SLICEL",placed CLBLL_X23Y86
SLICE_X39Y86 , ... D6LUT :PR1/data_i(0)_PROXY :#LUT :O6=A2 ...

Publications personnelles

1 Articles dans des revues avec comité de lecture

[Marq_R1] N. Marques, H. Rabah, E. Dabellani, and S.Weber, A novel framework for the design of adaptable reconfigurable regions for the placement of variable-sized ip cores, Embedded Systems Letters, IEEE, 2012. Soumis en juin 2012

[Marq_R2] F. Duhem, N. Marques, F. Muller, H. Rabaha, S. Weber, and P. Lorenzini, Multi-standards video adaptation using a fast reconfiguration manager, Microprocessors and Microsystems, 2012. Article en collaboration avec le LEAT de Nice

2 Articles de congrès internationaux avec actes

[Marq_Aci1] N. Marques, H. Rabah, E. Dabellani, and S. Weber, Effcient reconfigurable entropy coder for embedded multi-standards video adaptation, in Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Computer Society Conference on, june 2011, pp. 144 - 149.

[Marq_Aci2] M. Guarisco, E. Dabellani, N. Marques, H. Rabah, Y. Berviller, and S. Weber, An effcient vlsi implementation of h.264/avc intra-frame transcoder, in Electronics, Circuits and Systems (ICECS), 18 th IEEE International Conference on, dec. 2011, pp. 1 - 4.

[Marq_Aci3] N. Marques, H. Rabah, E. Dabellani, and S. Weber, Partially reconfigurable entropy encoder for multi standards video adaptation, in Consumer Electronics (ISCE), IEEE 15th International Symposium on, june 2011, pp. 492 - 496.

3 Communications sans actes

[Marq_C1] N. Marques, H. Rabah, E. Dabellani, Y. Berviller, and S. Weber, Architecture reconfigurable pour le transcodage vidéo mpeg appliqué à la conservation et l'archivage de contenus vidéo, in 5éme workshop Applications Médicales de l'Informatique, Amina 2010, Monastir, Tunisie.

[Marq_C2] E. Dabellani, N. Marques, H. Rabah, Y. Berviller, and S. Weber, Reconfiguration dynamique d'un analyseur syntaxique pour le transcodage et l'adaptation vidéo, Colloque national, GDR SOC-SIP 2010, Cergy, France.

[Marq_C3] M. Guarisco, N. Marques, E. Dabellani, Y. Berviller, H. Rabah, and S. Weber, Conception de sopc pour application multimédia, 11èmes journées pédagogiques, JPCNFM 2010, Saint Malo, France.

[Marq_C4] E. Dabellani, N. Marques, H. Rabah, Y. Berviller, and S. Weber, Utilisation des threads dynamiques pour la modélisation de la reconfiguration dynamique, Colloque national, GDR SOC-SIP 2011, Cergy, France.

4 Vulgarisation

[Marq_V1] N. Marques, Décodeur vidéo pour les maisons connectées, Présentation de poster pour les Doctoriales 2011.

[Marq_V2] N. Marques, Architecture adaptative basée sur les technologies de fpga reconfigurable dynamiquement pour le transcodage multimédia, Ecole d'hiver Francophone sur les Technologies de Conception des Systèmes embarqués Hétérogènes 2012.

Bibliographie

[Alt] ALTERA : Site web altera - http ://www.altera.com/. 9

[Ama06] H. AMANO : A survey on dynamically reconfigurable processors. *IEICE Transactions on Communications*, 89(12) :3179:39, 2006. 19, 20

[AMB] AMBA : Amba specification rev2.0. arm, 6, 8 ,70. 42

[Ard] ARDMAHN : Site web ardmahn : Architecture reconfigurable dynamiquement et methodologie pour auto-adaptation home networking - http ://ardmahn.org/. 5

[Ber92] P. BERTIN : *Memoires actives programmables : conception, realisation et programmation.* These de doctorat, These de doctorat, Universit ede Paris 07, 1992. 18

[BIR10] M. BYSTROM et M. de-Frutos-Lopez I. RICHARDSON, S. Kannangara : Dynamic replacement of video coding elements. *In Signal Processing : Image Communication*, volume 25, pages 303–313. ISSN 0923-5965, 2010. 89

[BKS00] K. BAZARGAN, R. KASTNER et M. SARRAFZADEH : Fast template placement for reconfigurable computing systems. *Design Test of Computers, IEEE*, 17(1):68 –83, jan-mar 2000. 3

[BKT11] C. BECKHOFF, D. KOCH et J. TORRESEN : The xilinx design language (xdl) : Tutorial and use cases. june 2011. 78

[BMB10] I. BELAID, F. MULLER et M. BENJEMAA : New three-level resource manage-

ment enhancing quality of offline hardware task placement on fpga. *International Journal of Reconfigurable Computing*, 2010:20, 2010. 4

[Car03] J.M.P. CARDOSO : On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures. *Computers, IEEE Transactions on*, 52(10):1362 – 1375, oct. 2003. 51

[CH02] Katherine COMPTON et Scott HAUCK : Reconfigurable computing : a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, juin 2002. 2

[CHW00] Timothy J. CALLAHAN, John R. HAUSER et John WAWRZYNEK : The garp architecture and c compiler. *Computer*, 33(4):62–69, avril 2000. 2, 51

[CRGM11] J.A. CLEMENTE, J. RESANO, C. GONZALEZ et D. MOZOS : A hardware implementation of a run-time scheduler for reconfigurable systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(7):1263 –1276, july 2011. 3

[DE97] Oliver DIESSEL et Hossam ELGINDY : Run-time compaction of fpga designs. Rapport technique, 1997. 3

[DNMR+12] F. DUHEM, and F. Muller N. MARQUES, H. RABAHA, S. WEBER et P. LORENZINI : Multi-standards video adaptation using a fast reconfiguration manager. *Microprocessors and Microsystems, Article en collaboration avec le LEAT de Nice*, 2012. 34

[DP05] K. DANNE et M. PLATZNER : A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. *In Field Programmable Logic and Applications, 2005. International Conference on*, pages 568 – 573, aug. 2005. 2

[DP06] Klaus DANNE et Marco PLATZNER : An edf schedulability test for periodic tasks on reconfigurable hardware devices. *SIGPLAN Not.*, 41(7):93–102, juin 2006. 2

[DPW99] D. DEMIGNY, M. PAINDAVOINE et S. WEBER : Architecture a reconfiguration dynamique pour le traitement temps reel des images. *Technique et Science de information Numero Special Architectures Reconfigurables*, 1999. 20

[EV62] G. ESTRIN et C. R. VISWANATHAN : Organization of a "fixed-plus-variable" structure computer for computation of eigenvalues and eigenvectors of real symmetric matrices. *J. ACM*, 9(1):41–60, janvier 1962. 2

[FPPR04] M. FRANZMEIER, C. POHL, M. PORRMANN et U. RUCKERT : Hardware accelerated data analysis. *In Parallel Computing in Electrical Engineering, 2004. PARELEC 2004. International Conference on*, pages 309 – 314, sept. 2004. 18

[GGJ+04] M. GOKHALE, P. GRAHAM, E. JOHNSON, N. ROLLINS et M. WIRTHLIN : Dynamic reconfiguration for management of radiation-induced faults in fpgas. *In Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 145, april 2004. 4

[HB06] M. HÜBNER et J. BECKER : Exploiting dynamic and partial reconfiguration for fpgas : toolflow, architecture and system integration. *In Proceedings of the 19th annual symposium on Integrated circuits and systems design*, SBCCI '06, pages 1–4, New York, NY, USA, 2006. ACM. 21, 80, 81

[HBB04] Michael HUEBNER, Tobias BECKER et Juergen BECKER : Real-time lut-based network topologies for dynamic and partial fpga self-reconfiguration. *In Proceedings of the 17th symposium on Integrated circuits and system design*, SBCCI '04, pages 28–32, New York, NY, USA, 2004. ACM. 33, 35, 37

[HH08] Chun-Hsian HUANG et Pao-Ann HSIUNG : Software-controlled dynamically swappable hardware design in partially reconfigurable systems. *EURASIP Journal on Embedded Systems*, 2008(1):231940, 2008. xiv, 43, 44, 45

[HKKP07a] J. HAGEMEYER, B. KELTELHOIT, M. KOESTER et M. POMNANN : A design

methodology for communication infrastructures on partially reconfigurable fpgas. pages 331 –338, aug. 2007. 55, 80, 81

[HKKP07b] Jens HAGEMEYER, Boris KETTELHOIT, Markus KOESTER et Mario PORRMANN : Design of homogeneous communication infrastructures for partially reconfigurable fpgas. pages 238–247, 2007. 55, 80, 81

[HSB06a] M. HUBNER, C. SCHUCK et J. BECKER : Elementary block based 2-dimensional dynamic and partial reconfiguration for virtex-ii fpgas. *In Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006. 4

[HSB06b] M. HUBNER, C. SCHUCK et J. BECKER : Elementary block based 2-dimensional dynamic and partial reconfiguration for virtex-ii fpgas. page 8 pp., april 2006. 20, 54

[JBGR10] A. JARA-BERROCAL et A. GORDON-ROSS : Vapres : A virtual architecture for partially reconfigurable embedded systems. *In Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 837 –842, march 2010. xiv, 46, 47

[Jov09] Slavisa JOVANOVIC : *Architecture reconfigurable de systeme embarque auto-organise*. These de doctorat, These de doctorat, UHP - Universite Henri Poincare, 2009. 43, 86, 88

[JS00] D. JAGGER et D. SEAL : Armarchitecture referencemanual. *In Pearson Education*, pages 819 –824, march 2000. 40

[JTW07] S. JOVANOVIC, C. TANOUGAST et S. WEBER : A hardware preemptive multitasking mechanism based on scan-path register structure for fpga-based reconfigurable systems. *In Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, pages 358 –364, aug. 2007. 87

[JTW08] S. JOVANOVIC, C. TANOUGAST et S. WEBER : A new high-performance scalable dynamic interconnection for fpga-based reconfigurable systems. *In Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, pages 61 –66, july 2008. 87

[JTWB07] S. JOVANOVIC, C. TANOUGAST, S. WEBER et C. BOBDA : Cunoc : A scalable dynamic noc for dynamically reconfigurable fpgas. *In Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 753 – 756, aug. 2007. 87

[JTWB09] S. JOVANOVIC, C. TANOUGAST, S. WEBER et C. BOBDA : A new deadlock-free fault-tolerant routing algorithm for noc interconnections. *In Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 326 –331, 31 2009-sept. 2 2009. 87

[KBT09] Dirk KOCH, Christian BECKHOFF et Jurgen TEICH : A communication architecture for complex runtime reconfigurable systems and its implementation on spartan-3 fpgas, 2009. 58

[KHT08] D. KOCH, C. HAUBELT et J. TEICH : Efficient reconfigurable on-chip buses for fpgas. *In Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, pages 287 –290, april 2008. 58

[LBM+06] P. LYSAGHT, B. BLODGET, J. MASON, J. YOUNG et B. BRIDGFORD : Invited paper : Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas. pages 1 –6, aug. 2006. Thilo (1). 30

[Liu08] Ting LIU : *Optimisation par synthese architecturale des methodes de partitionnement temporel pour les circuits reconfigurables*. These de doctorat, These de doctorat, UHP - Universite Henri Poincare, Novembre 2008. 18

[MPR+06] M. MURGIDA, A. PANELLA, V. RANA, M.D. SANTAMBROGIO et D. SCIUTO :

Fast ip-core generation in a partial dynamic reconfiguration workflow. *In Very Large Scale Integration, 2006 IFIP International Conference on*, pages 74 –79, oct. 2006. 44, 45

[MRDW11a] N. MARQUES, H. RABAH, E. DABELLANI et S. WEBER : Efficient reconfigurable entropy coder for embedded multi-standards video adaptation. *In Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 144 –149, june 2011. 20, 93

[MRDW11b] N. MARQUES, H. RABAH, E. DABELLANI et S. WEBER : Partially reconfigurable entropy encoder for multi standards video adaptation. *In Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, pages 492 –496, june 2011. 34, 93

[OV01] I. OUAISS et R. VEMURI : Hierarchical memory mapping during synthesis in fpga-based reconfigurable computers. *In Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 650 –657, 2001. 44

[OWTK10] A. OETKEN, S. WILDERMANN, J. TEICH et D. KOCH : A bus-based soc architecture for flexible module placement on reconfigurable fpgas. pages 234 –239, 31 2010-sept. 2 2010. xiv, 43, 44, 58, 59, 80, 81

[PAKM09] T. PIONTECK, C. ALBERCHT, R. KOCH et E. MAEHLE : Adaptive communication architectures for runtime reconfigurable system-on-chips. 18(2):275–289, 2009. 55, 80, 81

[PB99] K.M.G. PURNA et D. BHATIA : Temporal partitioning and scheduling data flow graphs for reconfigurable computers. *Computers, IEEE Transactions on*, 48(6): 579 –590, jun 1999. 19

[PKA06] Thilo PIONTECK, Roman KOCH et Carsten ALBRECHT : Applying partial re-

configuration to networks-on-chips. *In FPL*, pages 1–6, 2006. xiv, 55, 57, 80, 81

[PKAM09] T. PIONTECK, R. KOCH, C. ALBERCHT et E. MAEHLE : A design technique for adapting number and boundaries of reconfigurable modules at runtime. *Hindawi*, 2009. xiv, 55, 56, 80, 81

[SF08] M.L. SILVA et J.C. FERREIRA : Generation of partial fpga configurations at run-time. pages 367 –372, sept. 2008. 55

[Shi09] Chia-Cheng Lo ; Shang-Ta Tsai ; Ming-Der SHIEH : A reconfigurable architecture for entropy decoding and idct in h.264. *In VLSI Design, Automation and Test ; VLSI-DAT '09. International Symposium on*, pages 279–282, 2009. 89

[STM] STMICROLELECTRONIQUE : Stbus communication system : concept and definition. stmicrolelectronique internal document. 8, 37, 76. 42

[Tsa09] Ming-Ju Wu ; Yi-Tseng Chen ; Chun-Jen TSAI : Hardware-assisted syntax decoding model for software avc/h.264 decoders. *In Circuits and Systems. ISCAS*, 2009. 89

[VBR+96] J.E. VUILLEMIN, P. BERTIN, D. RONCIN, M. SHAND, H.H. TOUATI et P. BOUCARD : Programmable active memories : reconfigurable systems come of age. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 4(1):56 –69, march 1996. 18, 72

[WK02] Grant WIGLEY et David KEARNEY : The management of applications for reconfigurable computing using an operating system. *Aust. Comput. Sci. Commun.*, 24(3):73–81, janvier 2002. 3

[WNP04] Herbert WALDER, Samuel NOBS et Marco PLATZNER : Xf-board : A prototyping platform for reconfigurable hardware operating systems. *In In Proceedings*

of the 4th International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA). CSREA. Press, 2004. 54

[WP04] Herbert WALDER et Marco PLATZNER : A runtime environment for reconfigurable hardware operating systems. pages 831–835, 2004. xiv, 53, 54, 80, 81

[WSP03] Herbert WALDER, Christoph STEIGER et Marco PLATZNER : Fast online task placement on fpgas : Free space partitioning and 2d-hashing. *Parallel and Distributed Processing Symposium, International*, 0:178b, 2003. 80, 81

[XDN09] Yan XIAO, Zhenhua DUAN et Pengcheng NIE : An efficient algorithm for finding empty space for reconfigurable systems. *In Theoretical Aspects of Software Engineering, 2009. TASE 2009. Third IEEE International Symposium on*, pages 36 –43, july 2009. 4

[XE] Inc XILINX et Emi ETO : Difference-based partial reconfiguration. 32

[Xila] XILINX : Site web xilinx - http ://www.xilinx.com/. 9

[Xilb] Inc XILINX : Development system reference guide, chapter 4, modular design. 2002. 29

[Xilc] Inc XILINX : Ise design suite 13 :release notes guide ug631 (v 13.2) july 6, 2011. 74

[Xild] Inc XILINX : Ug208 early access partial reconfigurable user guide, 2006. 32

[Xile] Inc XILINX : Virtex 5 fpga configuration user guide, aug. 2010. available online. 28

[Xilf] Inc XILINX : Virtex 6 fpga configuration user guide, nov. 2010. available online. 28

[XLP] Inc XILINX, Davin LIM et Mike PEATTIE : Two flows for partial reconfiguration. 30

[XM] Inc XILINX et Mounir MAAREF : Creating an opb ipif-based ip and using it in edk ; xapp967 (v1.1) february 26, 2007. 45

[YCC01] YUNG-CHI et Liang-Gee Chen. CHANG, Rlao-Chieh Chang : Design and implementation of a bitstream parsing coprocessor for mpeg-4 video system-on-chip solution. *In* International SYMPOSIUM, éditeur : *VLSI Technology, Systems, and Applications,*, pages 188–191, 2001. 89

Résumé

Les architectures reconfigurables à base de FPGA sont capables de fournir des solutions adéquates pour plusieurs applications vu qu'elles permettent de modifier le comportement d'une partie du FPGA pendant que le reste du circuit continue de s'exécuter normalement. Ces architectures, malgré leurs progrès, souffrent encore de leur manque d'adaptabilité face à des applications constituées de tâches matérielles de taille différente. Cette hétérogénéité peut entraîner de mauvais placements conduisant à une utilisation sous-optimale des ressources et par conséquent une diminution des performances du système.

La contribution de cette thèse porte sur la problématique du placement des tâches matérielles de tailles différentes et de la génération efficace des régions reconfigurables. Une méthodologie et une couche intermédiaire entre le FPGA et l'application sont proposées pour permettre le placement efficace des tâches matérielles de tailles différentes sur des partitions reconfigurables de taille prédéfinie. Pour valider la méthode, on propose une architecture basée sur l'utilisation de la reconfiguration partielle afin d'adapter le transcodage d'un format de compression vidéo à un autre de manière souple et efficace. Une étude sur le partitionnement de la région reconfigurable pour les tâches matérielles de l'encodeur entropique (CAVLC / VLC) est proposée afin de montrer l'apport du partitionnement. Puis une évaluation du gain obtenu et du surcoût de la méthode est présentée.

Mots-clés: *Architecture auto-adaptatives, méthodologie de partitionnement, reconfiguration partielle, FPGA, placement de tâches matérielles*

Abstract

FPGA-based reconfigurable architectures can deliver appropriate solutions for several applications as they allow for changing the performance of a part of the FPGA while the rest of the circuit continues to run normally. These architectures, despite their improvements, still suffer from their lack of adaptability when confronted with applications consisting of variable size material tasks. This heterogeneity may cause wrong placements leading to a sub-optimal use of resources and therefore a decrease in the system performances.

The contribution of this thesis focuses on the problematic of variable size material task placement and reconfigurable region effective generation. A methodology and an intermediate layer between the FPGA and the application are proposed to allow for the effective placement of variable size material tasks on reconfigurable partitions of a predefined size. To approve the method, we suggest an architecture based on the use of partial reconfiguration in order to adapt the transcoding of one video compression format to another in a flexible and effective way. A study on the reconfigurable region partitioning for the entropy encoder material tasks (CAVLC / VLC) is proposed in order to show the contribution of partitioning. Then an assessment of the gain obtained and of the method additional costs is submitted.

Keywords: *Self-adaptive architecture, partitioning methodology , partial reconfiguration, FPGA, placement of hardware tasks*